UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

LOCALIZACIÓN MEDIANTE DIFFERENTIAL EVOLUTION DE UN ROBOT MÓVIL

PROYECTO FIN DE CARRERA

INGENIERÍA TÉCNICA INDUSTRIAL
ESPECIALIDAD ELECTRÓNICA INDUSTRIAL

AUTOR: EDUARDO RIVAS GONZÁLEZ
TUTOR: Dr. SANTIAGO GARRIDO BULLÓN

SEPTIEMBRE 2011

A mi familia

AGRADECIMIENTOS

Este proyecto es el resultado de un intenso periodo de esfuerzo y de trabajo. Durante este tiempo, son muchas las personas que, de una u otra forma, me han ayudado a alcanzar la meta, por ello, quiero expresarles mi más sincera gratitud.

En primer lugar agradezco a Santiago Garrido, mi tutor, la oportunidad que me brindó en su día de embarcarme en este interesante proyecto y, sobretodo, la paciencia mostrada y la gran ayuda prestada durante todo el recorrido.

También agradezco a Luis Moreno, su disponibilidad e interés en las ocasiones en las que he precisado de su ayuda.

En un aspecto más personal, agradecérselo de forma especial a mis padres, Luis y Araceli, a mi hermana, Araceli, y a mi hermano, Luis. Se hace difícil encontrar las palabras exactas que expresen mi grado de gratitud, lo que sí tengo claro es que soy lo que soy gracias a ellos, entre otras cosas, por todo el cariño, apoyo y los buenos consejos que me han dedicado a lo largo de mi vida. Tampoco puedo olvidarme de Silvia, gracias por haber estado en todo momento a mi lado y creer siempre en mí. Por último, gracias a todos los amigos y compañeros que surgieron durante estos años en la Universidad.

RESUMEN

Los primeros robots operaban en entornos especialmente preparados para ellos. Cada componente de su espacio de trabajo se encontraba situado en una posición y orientación predefinidas, de modo que el robot conocía *a priori* y con exactitud el escenario donde se encontraba. En la actualidad muchas de las aplicaciones requieren que los robots tengan rasgos de autonomía como son la capacidad para identificar mediante sus sensores las características que tiene el entorno o la capacidad para auto-localizarse dentro del mismo.

En los últimos años se han propuesto una gran variedad de técnicas de posicionamiento de robots móviles [1]. Su planteamiento varía considerablemente en función del medio en el cual se mueve el robot, del conocimiento que se posea tanto del entorno como de la tarea a realizar, y del conjunto de sensores disponible. En general, la posición puede determinarse a través de sensores, tanto internos como externos [2].

A pesar de que los resultados obtenidos en el campo de la auto-localización de robot móviles son abundantes y muy significativos, aún existen problemas por resolver, sobre todo, los vinculados al uso de sensores imprecisos y de bajo costo, ya que la información obtenida con estos sensores resulta poco fiable en el momento de integrarla a las distintas aplicaciones en este tipo de sistemas.

El presente trabajo se centra en la implantación de un sistema de localización absoluta basado en técnicas evolutivas, en un robot real de carácter educativo y comercial dotado con sensores de bajo costo.

ABSTRACT

The first robots operating in environments specially prepared for them. Each component of your workspace was located in a predefined position and orientation so that the robot known a priori and accurately the stage where he was. At present, many applications require that robots have autonomous features such as the ability to identify by its sensors the features that the environment or the ability to self-localized inside it.

In recent years we have proposed a variety of positioning techniques for mobile robots [1]. His approach varies considerably depending on the medium in which the robot moves, the existing knowledge of both the environment and the task at hand, and the set of available sensors. In general, the position can be determined through sensors, both internal and external [2].

Although the results obtained in the field of self-localization of mobile robots are numerous and very significant, there are still unresolved problems particularly those related to the use of inaccurate sensors, low cost, since the information gained from these sensors is unreliable at the time to integrate the various applications on such systems.

This work focuses on the implementation of an absolute positioning system based on evolutionary techniques in a real robot designed for educational and commercial uses equipped with low cost sensors.

ÍNDICE

1. INTRODUCCIÓN

Uno de los campos más prometedores de la robótica móvil, es el de otorgar autonomía a vehículos terrestres con el objetivo de reducir la intervención de supervisores humanos en entornos de diferente complejidad, desde tareas rutinarias a trabajos en ambientes tóxicos o climatológicamente adversos.

Teniendo en cuenta esto, no se puede incluir dentro de esta categoría ningún tipo de máquina teleoperada ni vehículos autoguiados, ya que unos dependen del control humano y otros se limitan a seguir caminos preestablecidos o a actuar de forma repetitiva.

La necesidad de movimiento no teledirigido en robótica se aprecia claramente en sistemas que desarrollan su operación en entornos remotos.

Se entiende el control de un robot como la monitorización continua del mismo, es decir, conocer en cada momento la posición y orientación respecto un cero u origen. A este concepto se le denomina localización.

El conocimiento en tiempo real de la posición precisa del robot móvil dentro de un entorno, nos permite optimizar la planificación de las trayectorias ante situaciones imprevistas, llevando a cabo las tareas impuestas con precisión y eficiencia.

El giro de los elementos motrices de un vehículo móvil, independientemente de que sean ruedas o cadenas, puede proporcionar información suficiente como para determinar su posición. En estas situaciones se dice que la localización está basada en la odometría.

Sin embargo, existen errores asociados a deslizamientos, fricciones o la propia orografía del terreno que son acumulables. De este modo, llegará un momento en el que el sistema no pueda determinar con el grado de exactitud necesario la posición del robot.

Por tanto, se precisa de un sistema alternativo de localización, que sin omitir los resultados provenientes de las mediciones odométricas, corrijan la posición del vehículo y localicen el mismo con la mayor exactitud posible. Un proceso de localización, el cuál, a través de la información proveniente de los sensores externos, sea capaz de determinar la posición y orientación ocupada por el robot móvil.

Existen una gran variedad de técnicas de localización de robots móviles. El enfoque de cada una de ellas va determinado en función de las características previas, como el conjunto de sensores disponible, el medio en el cuál se mueve el robot o el grado de conocimiento del entorno y de las tareas a realizar.

1.1. OBJETIVO

El presente trabajo se ocupa, fundamentalmente, de dos aspectos. Por una parte, se centra en el desarrollo de subrutinas que nos permitan el control de un robot autónomo, que mediante sus sensores será capaz de percibir y modelar el entorno, a fin de tomar decisiones en la planificación de estrategias de movimiento con el objetivo de esquivar cualquier obstáculo que se encuentren en su trayectoria.

Para nuestro caso particular, el robot partirá desde un punto inicial no conocido, será capaz de encontrar la salida en un entorno laberíntico del cual no necesita tener información previa.

El robot utilizado es un LEGO MINDSTORMS NXT. Se programará utilizando la toolbox *RWTH - Mindstorms NXT* de MATLAB, desarrollado por la Universidad RWTH de Aachen, Alemania. La elección de MATLAB como soporte para la programación dota a este robot, de índole comercial, de unas altas prestaciones, ya que queda a su servicio todo el potencial de cálculo del mejor software matemático.

En segundo lugar, el objetivo perseguido es la implantación de un sistema de localización, que permitirá al vehículo conocer su posición y orientación. Para este acometido, el entorno debe conocerse *a priori*, por lo que se establecerá un diseño de carácter fijo para el laberinto.

Para la conclusión de este objetivo se hace uso de una aplicación desarrollada en el Departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid. Dicha aplicación está basada en la implementación de un algoritmo evolutivo diferencial, concretamente, nos referimos al algoritmo *Differential Evolution* de Rainer Storn y Kenneth Price.

A pesar de que el problema de la localización se ha estudiado mediante muchos otros métodos, el uso de algoritmos genéticos proporciona, entre otras aplicaciones, un potente mecanismo de búsqueda paralela muy eficiente en términos de localización.

1.2. ESTRUCTURA DEL PROYECTO

El proyecto se encuentra estructurado en los siguientes capítulos:

- **Capítulo 1:** Este capítulo.

- **Capítulo 2:** En este capítulo introduciremos qué se entiende por un robot móvil e incidiremos en la explicación de las técnicas más empleadas en la localización.

- **Capítulo 3:** Se da al lector, una idea del área de la computación evolutiva, centrando nuestro interés en las características propias de los Algoritmos Genéticos y en el Algoritmo de Evolución Diferencial y su aplicación como métodos de localización.

- **Capítulo 4:** Describiremos, de forma detallada, las propiedades de hardware que compone el robot móvil empleado, así como las aplicaciones existentes para su programación.

- **Capítulo 5:** En él se encuentran explicados detenidamente las subrutinas desarrolladas empleadas para el control, adquisición y tratamiento de datos del robot móvil, y su relación con en el algoritmo de localización.

- **Capítulo 6:** Se presentan los resultados experimentales obtenidos durante la ejecución del algoritmo de Evolución Diferencial definido en apartados anteriores.

- **Capítulo 7:** En este apartado, se muestran las conclusiones obtenidas basadas en los resultados obtenidos, así como las posibles líneas de investigación a seguir.

- **Capítulo 8:** Se recoge la relación de recursos bibliográficos y electrónicos utilizados.

- **Capítulo 9:** Se presenta el código fuente desarrollado para el control del robot móvil.

2. ESTADO DEL ARTE

El problema de la localización de robots móviles consiste en contestar, desde el punto de vista del robot, a la pregunta: ¿Dónde estoy? Esto significa que el robot, valiéndose de sus sensores, debe encontrar su posición relativa al entorno donde se encuentra. El problema general de la localización consta de un cierto número de instancias que se pueden clasificar según su grado ascendente de dificultad: seguimiento de posición (Tracking), posicionamiento global (Global localization), recuperación ante transferencias intempestivas (Kinapping problem) y localización en ambientes dinámicos.

La localización del robot resulta clave en el diseño de robots verdaderamente autónomos. Si un robot no sabe dónde está, probablemente resulte difícil determinar qué hacer a continuación. Con el fin de localizarse a sí mismo, un robot tiene acceso a la información relativa y absoluta proporcionada por sus sensores internos y externos que nos permiten conocer, respectivamente, la distancia recorrida y las características del entorno del robot. Teniendo en cuenta esta información, el robot tiene que determinar su posición con la mayor precisión posible.

Esta adquisición de información se ve afectada por la imprecisión propia de los dispositivos sensoriales por lo que es preciso combinarlas de forma óptima con diferentes sistemas de localización con el fin de estimar, con la mayor precisión posible, la posición real del robot.

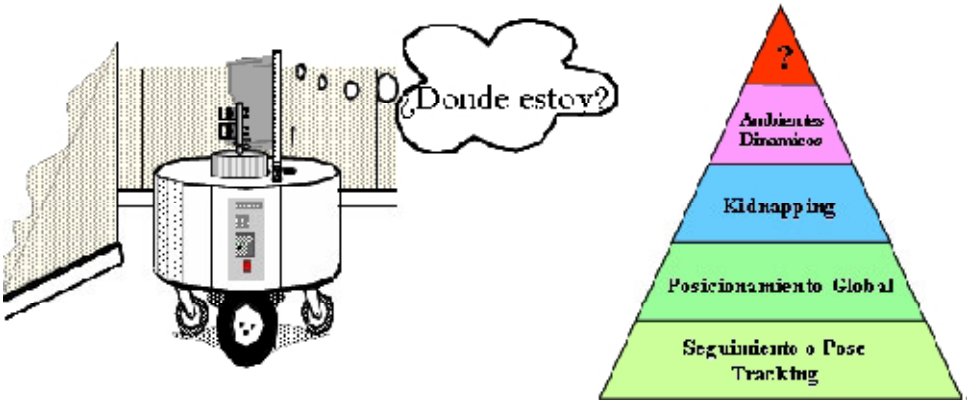


Figura 2.1.: Clasificación jerárquica de la localización de robots móviles

2.1. ROBOTS MÓVILES AUTÓNOMOS

La robótica es la ciencia y la tecnología de los robots. Combina diversas disciplinas como son: la mecánica, la electrónica, la informática, la inteligencia artificial y la ingeniería de control.

El término robot se popularizó con el éxito de la obra *RUR (Robots Universales Rossum)*, escrita por Karel Capek en 1920. En la traducción al inglés de dicha obra, la palabra checa *"robota"*, que significa trabajos forzados, fue traducida al inglés como *robot*. Desde entonces, se ha avanzado mucho en investigación en la robótica, dónde se ha tornado el concepto inicial de robots estáticos y no autónomos al predominio actual de la movilidad y la autonomía.

2.1.1. MOVILIDAD Y AUTONOMÍA

La movilidad de los robots representa el grado en que éstos son capaces de moverse libremente. El primer uso práctico de los robots fue en entornos industriales con los llamados robots manipuladores [3]. Éstos tienen la tarea de fabricar productos como los automóviles. Están programados de tal manera, que puedan repetir la misma secuencia de acciones una y otra vez, más rápido, más barato y más preciso que los seres humanos. Por lo general, consisten en un brazo móvil fijado a un punto en el suelo, con la habilidad de montar o fabricar piezas diferentes. Pueden moverse con destreza en torno a este punto fijo, sin embargo, no son capaces de moverse libremente.

Si queremos que los robots puedan realizar tareas cotidianas como la limpieza del piso, prestar ayuda a personas dependientes o que puedan ser utilizados para explorar los ambientes que son difíciles o

peligrosos para los seres humanos como los radiactivos o tóxicos, trabajos submarinos o misiones a los planetas y otros lugares en el espacio, tendrán que poseer un grado de movilidad mucho mayor al de los robots manipuladores. Es decir, tienen que ser capaces de moverse libremente en el entorno que le rodea en el desempeño de sus funciones.

La autonomía de los robots depende de hasta qué punto un robot se basa en el conocimiento previo o la información del entorno para lograr su cometido. Se pueden dividir en tres tipos de autonomía: [4].

- **No Autónomos:** Son completamente dirigidos de forma remota por seres humanos. La inteligencia desarrollada por estos robots consiste en interpretar los comandos recibidos de los controladores humanos.

- **Semi-Autónomos:** Pueden navegar por sí mismos pero existe cierto grado de intervención del hombre. Un ejemplo de semi-autonomía se logra mediante el ajuste del entorno dónde todo está adaptado de tal manera que el robot pueda moverse con seguridad. Otra posibilidad consiste en proporcionar el mapa del entorno dónde van a desarrollar las tareas [5,6].

- **Autónomos:** No requieren la interacción humana para cumplir con sus tareas. Los vehículos robots totalmente autónomos son capaces de desarrollar acciones y movimientos inteligentes, sin necesidad de ninguna orientación externa [3].

2.2. SENSORIZACIÓN EN ROBÓTICA MÓVIL

Muy estrechamente ligado a la autonomía de un robot está la capacidad de percibir el entorno y actuar sobre el mismo. Por medio de los sensores el robot obtiene datos sobre su entorno (sensación) y luego los procesa para su utilización (percepción). De esta forma, los sensores representan para el robot la fuente de datos sobre los cambios, tanto en su entorno (distancia a objetos, luz ambiental) como en sí mismo (nivel de la baterías, consumo de los motores, pulsos de codificadores, etc.).

En robótica móvil se usa una gran variedad de sensores que miden distintas magnitudes. En general, se puede clasificar en dos grupos: sensores internos y sensores externos. Los sensores internos son aquellos que no interaccionan con el entorno del robot. Su función es suministrar datos relacionados a las variables cinemáticas del robot y de cómo varían en función del accionamiento de los motores. Los sensores internos más característicos

son: potenciómetros, tacómetros, codificadores ópticos rotativos, giroscopios y acelerómetros.

Los sensores externos son sistemas que interaccionan con el entorno para cuantificar alguna variable perteneciente al mismo. Entre los sensores externos cabe destacar:

- **Ultrasonidos:**

  Son ampliamente utilizados en aplicaciones de modelado del entorno. Permiten la medida de distancias a partir del tiempo transcurrido desde la emisión de un pulso de ultrasonido hasta que éste es recibido nuevamente por el receptor. La precisión obtenida puede ser de un centímetro pero las medidas dependen de factores exteriores como la temperatura ambiente, movimientos del aire y fuentes acústicas de alta frecuencia (máquinas rotativas). Tienen un alcance entre 40 cm y 10 m.

- **Infrarrojos:**

  Tienen mayor precisión (±2mm) pero menor alcance que los sensores de ultrasonidos (entre 10 cm y 80 cm) por lo que son útiles, especialmente, para maniobra de aproximación y de navegación por evitación de obstáculos.

- **Láser:**

  Pueden ser utilizados para detectar medidas de distancia a objetos opacos, así como para la localización mediante triangulación, determinando la distancia a puntos conocidos del entorno. Poseen buena precisión y velocidad de adquisición de datos pero, en contrapartida, presentan un costo generalmente elevado.

- **Visión:**

  Las cámaras son ampliamente usadas en sistemas de localización y construcción de mapas del entorno. Los sistemas de visión estereoscópica, consisten en un par de cámaras que, además de conseguir la detección de contornos, también permiten obtener, de forma aproximada, la distancia a objetos a partir de la divergencia de ambas cámaras.

En la robótica móvil, los sensores que más se utilizan son los orientados a resolver el problema de la determinación de la posición. Enmarcado dentro de esta problemática, en [7, 8] se lleva a cabo una revisión de varias de las tecnologías y metodologías utilizadas en el ámbito de sensores y procesamiento de la información proporcionada por los mismos.

2.3. LOCALIZACIÓN ODOMÉTRICA

El proceso más básico para la localización de un robot móvil se basa en el modelo cinemático del sistema de propulsión. La posición de un robot tipo diferencial puede ser estimada a partir de las ecuaciones geométricas que surgen de la relación entre los componentes del sistema de propulsión y de la información de los codificadores rotativos que usualmente llevan acoplados a sus ruedas. Este procedimiento es conocido como odometría.

La palabra "odometría" se compone por las palabras griegas *hodos* ("viajar", "trayecto") y *metron* ("medida"). Los robots móviles usan la odometría para estimar, y no determinar, su posición relativa a su localización inicial. Este método proporciona una buena precisión a corto plazo, es barata de implantar, y permite tasas de muestreo muy altas. Sin embargo, la idea fundamental de la odometría es la integración de información incremental del movimiento a lo largo del tiempo, lo cual conlleva una inevitable acumulación de errores. En concreto, la acumulación de éstos en la orientación, causa grandes errores en la estimación de la posición, los cuales van aumentando proporcionalmente con la distancia recorrida por el robot.

A pesar de estas limitaciones, muchos investigadores coinciden en que la odometría es una parte importante del sistema de navegación de un robot, siempre que se utilice con un sistema de corrección, que actúe de forma periódica o en determinados puntos críticos del camino.

Los errores asociados al sistema odométrico pueden clasificarse en errores sistemáticos y errores no sistemáticos.

- **ERRORES SISTEMÁTICOS**

  Son independientes del entorno del robot e inherentes a la cinemática del mismo, por lo que siempre están presentes. Entre los más usuales se encuentran:

  - Diámetros de las ruedas diferentes, normalmente debido al desgaste.
  - El diámetro nominal de las ruedas difiere del diámetro real.
  - Mal alineamiento de las ruedas.
  - Límites del encoder: baja resolución y/o baja frecuencia de muestreo.

- **ERRORES NO SISTEMÁTICOS**

  Este tipo de error depende directamente del entorno que rodea al robot. Entre los más frecuentes podemos encontrar:

  - Desplazamiento en suelos desnivelados.

- Desplazamiento sobre objetos inesperados que se encuentren en el suelo.
- Deslizamiento de las ruedas debido a:
  - ✓ Superficie deslizante.
  - ✓ Sobre-aceleración.
  - ✓ Derrapes (debidos a una rotación excesivamente rápida).
  - ✓ Inexistencia de punto de contacto con la superficie.
- Fuerzas externas debidas a iteraciones con cuerpos externos.
- Fuerzas internas causadas, por ejemplo, al mal estado de las ruedas guías.

Una clara distinción entre errores sistemáticos y no sistemáticos es de gran importancia a la hora de reducir los errores en la odometría. Los sistemáticos son especialmente graves, porque se acumulan constantemente, y, al cabo de un tiempo de navegación, la información ya no es válida, de forma que habrá que corregir la posición. El mayor inconveniente de los errores no sistemáticos es que pueden aparecer inesperadamente y pueden causar errores muy grandes en la estimación de la posición.

De todos los errores, sólo aquellos debidos a imperfecciones en el diseño mecánico y sensorial de vehículo se mantienen constantes y pueden ser eliminados en el proceso de calibrado, utilizando técnicas más complejas como la navegación inercial basada en acelerómetros para medir las variaciones de velocidad, y giroscopios para la orientación, donde la precisión obtenida es muy superior, pero también conlleva un coste superior [9].

En esta situación, es necesario un sistema alternativo de localización, que teniendo en que cuenta los resultados odométricos obtenidos, corrija la posición del robot y la localización goce de un grado de exactitud mayor.

2.4. NAVEGACIÓN INERCIAL

Los sistemas de navegación inercial conocidos como INS (del inglés "Inertial Navigation Sistem") surgen de la necesidad de controlar la posición de vehículos en espacios de tres dimensiones. Por este motivo, estos sistemas de posicionamiento suelen aplicarse en vehículos espaciales, submarinos, barcos o aeronaves.

Los INS estiman la posición y orientación del vehículo empleando acelerómetros y giroscopios, con los que se genera un sistema inercial coordinado. El número de sensores empleados va en función del número de dimensiones que se necesite, así pues, para un movimiento tridimensional se utilizarán tres acelerómetros y tres giroscopios, de los que obtendremos un vector de coordenadas *[x(t), y(t), z(t)]* y tres ángulos que definen el movimiento del vehículo.

Con los datos obtenidos de los acelerómetros se calcularán la velocidad lineal y la posición mediante la primera y segunda derivada, respectivamente. Por su parte, los ángulos entregados por los giroscopios aportan información sobre la orientación del vehículo respecto a un sistema dado. Estos ángulos pueden ser de alabeo, cabeceo y guiñada según el sistema coordinado del vehículo o pueden ser los ángulos de Euler, si se define un sistema coordenado global. La elección de un sistema coordenado u otro dependerá de la actividad realizada.

Los sistemas inerciales son más indicados que los odométricos, sobretodo si buscamos alta precisión en las actividades realizadas por el robot y no es posible disponer de información del entorno, ya que éstos muestran mayor fiabilidad al no verse afectados por la interacción con superficies irregulares o por movimientos accidentales, como pueden ser colisiones. Sin embargo, no resuelven el problema de los errores acumulados y presentan un elevado coste y menor robustez, siendo necesarias operaciones de calibrado con una alta frecuencia.

2.5. MODELOS DE ENTORNO

Un elemento fundamental de la localización es el modelo de representación del entorno. Un modelo o mapa del entorno es una abstracción con la que se representan únicamente aquellas características del entorno que se consideran útiles para la navegación o la localización del robot. Durante el modelado, se desechan las características que requieran de un elevado nivel de detalle, debido a que pueden ser demasiados variables o no pueden ser detectadas con fiabilidad por los sensores.

El modelo del entorno representa un elemento fundamental para la localización del robot. En general, los algoritmos de localización suelen comparar las lecturas obtenidas por los sensores del robot con el modelo del entorno, actualizando la posición del robot de forma acorde con el resultado de esta comparación.

Existen dos formas fundamentales de modelado de entorno: modelos topológicos y modelos métricos, éstos últimos, a su vez, se pueden dividir en modelos de rejilla y en modelos geométricos. En los siguientes apartados vamos a profundizar en las características de cada uno de ellos.

2.5.1. MAPAS TOPOLÓGICOS

Los mapas topológicos se basan en representar las características esenciales del entorno percibidas por el robot móvil utilizando un grafo

como un modelo de alto nivel, definido por nodos y ramas. Los nodos se utilizan para representar lugares del entorno de marcada importancia que poseen características sensoriales distinguibles de forma absoluta, o respecto a sus nodos vecinos y las interconexiones o ramas que representan el trayecto que comunican los nodos entre sí.

Los nodos corresponden a la unidad elemental de localización, de manera que toda una zona geométrica del mapa real se representa por un único lugar. A partir del mapa topológico no es posible distinguir localizaciones más finas que las representadas por los lugares.

En la siguiente figura podemos ver la propuesta de mapas topológicos de Kuipers [10].

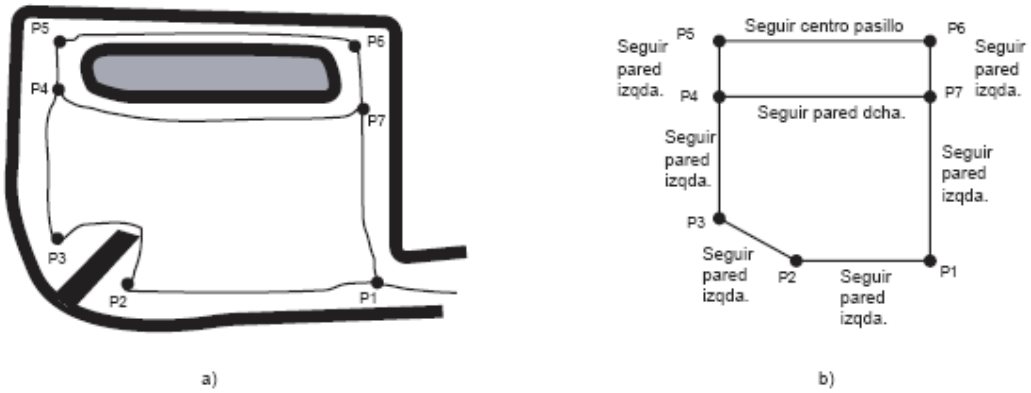


Figura 2.2.: Ejemplo de mapa topológico de Kuipers

Los nodos corresponden a puntos distintivos del entorno y las ramas a los trayectos recorridos por el robot. Una posición del entorno correspondiente a un nodo debe distinguirse localmente de su vecindad mediante algún criterio definible en términos de percepción sensorial. En los experimentos realizados por Kuipers, la función de distinción calcula el número de objetos cercanos que se encuentran a igual distancia del robot.

Los mapas topológicos han sido utilizando con múltiples variantes. Con frecuencia son extendidos agregando alguna información métrica, típicamente, incluyendo las longitudes estimadas de las trayectorias entre los nodos y la orientación de dichas trayectorias.

Entre las ventajas de los mapas topológicos están que permiten una planificación eficiente y una baja complejidad del espacio. En cuanto a las desventajas, cabe mencionar la dificultad para representar zonas abiertas donde el alcance limitado de los sensores obtiene información ambigua.

2.5.2. MAPAS MÉTRICOS

Representan las propiedades métricas o las coordenadas de los objetos del entorno (áreas, distancias, tamaño, localización, orientación, etc.). Este tipo de represtación suele realizarse en el mismo sistema de coordenadas 2D en el cual se representa al robot, lo que facilita la fusión de los datos propioceptivos y externoceptivos que se obtienen mediante los sensores del robot. Existen dos tipos de representación en este sentido, los mapas geométricos y los mapas de ocupación.

- **MAPA GEOMÉTRICO**

  Los modelos geométricos definen el entorno mediante sus características geométricas: puntos, líneas, arcos de círculo, etc. Estos mapas representan numéricamente las coordenadas y propiedades de los objetos del entorno donde se mueve el robot.

  La principal ventaja de estos modelos es que, utilizados juntos a un modelo de sensor eficaz, es posible simular los datos que los sensores del robot obtendrían en cualquier posición del entorno. Esto nos permite la comparación entre los datos percibidos por el robot y los datos que se obtendrían en posiciones candidatas, calculándose la probabilidad asociada a cada posición.

  Existe una amplia variedad de modelos geométricos del entorno. Uno de los más extendidos es el que describe el entorno a través de un conjunto de características geométricas (segmentos, esquinas, etc.) y una serie de relaciones entre ellas (distancia, posición, etc.) [11, 12].

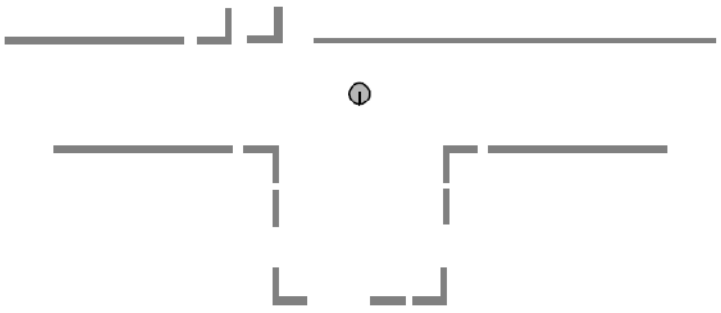


Figura 2.3.: Ejemplo de modelo geométrico del entorno. Las características geométricas usadas en el modelo son: esquinas, aristas y segmentos de rectas.

Otros enfoques definen el entorno como un mapa tipo CAD, que refleja los distintos elementos a modelar, sus dimensiones y posiciones [13, 14]. Cuánto mas detallado sea el modelo CAD mayor calidad tendrán las simulaciones de las lecturas de los sensores del robot en las posiciones candidatas.

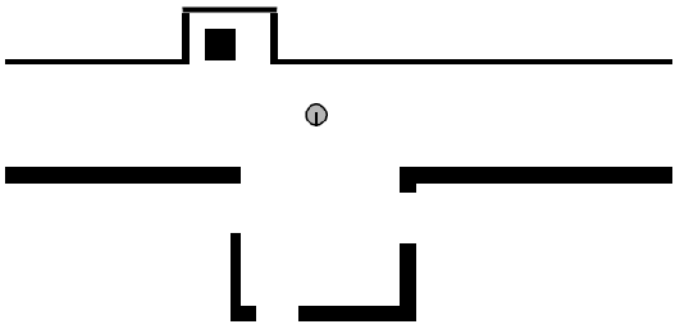


Figura 2.4.: Ejemplo del modelo CAD del entorno.

Por último, otros trabajos [15, 16] proponen utilizar como modelo del entorno los propios datos percibidos por los sensores del robot, aplicándoles el mínimo tratamiento posible como puede ser la corrección odométrica.

Figura 2.5.: Ejemplo modelo sensorial del entorno, en el que el entorno es el mismo que el modelado por las figuras 2.3 y 2.4.

- **MAPA DE OCUPACIÓN**

También conocido como mapa de rejilla. Consiste en dividir el entorno del mapeado en celdas a las que se le asigna una probabilidad de ocupación. Esta representación discretiza el entorno en celdas de igual dimensión que contendrán información acerca de si el área que ella representa está ocupada o no. El método más simple consiste en dividir el espacio usando una rejilla cuadrada. Cada una de estas celdas mantiene una probabilidad de que la zona de entorno que ella representa esté ocupada.

La principal ventaja de los mapas de ocupación es su generalidad: no se hace ninguna suposición acerca del tipo de objetos presentes en el entorno. Las celdas pueden representar cualquier objeto. Comúnmente, las celdas sólo guardan la probabilidad de ocupación de la celda y, en este caso, se habla de un mapa de ocupación probabilista.

La principal desventaja es que la resolución o fidelidad es limitada por el tamaño de la celda y la representación requiere la misma cantidad de almacenamiento aun si gran parte del entorno está libre u ocupado. La cantidad de celdas puede llegar a representar un problema, especialmente si se consideran ambientes bidimensionales grandes, ya que se requerirán grandes recursos de almacenamiento.

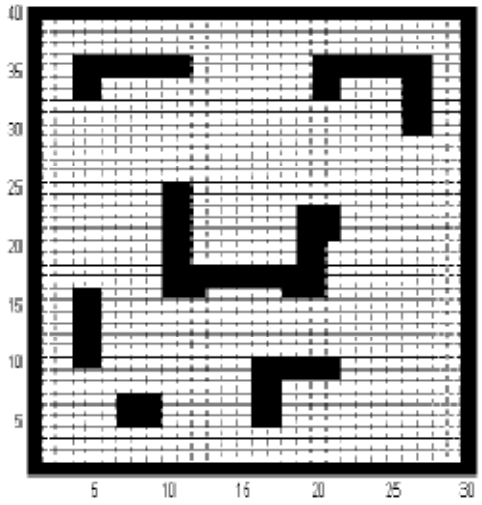


Figura 2.6.: Ejemplo mapa de rejilla

2.6. MÉTODOS DE LOCALIZACIÓN

Los métodos de localización presentan como objetivo compensar el error relativo a la odometría que tiene lugar durante la navegación. Para ello se hace uso de la información proveniente de los sensores.

La manera de clasificar estos métodos es muy variada y atiende a diferentes criterios. Una posible agrupación es la clasificación de estos métodos en métodos locales o globales, en métodos activos o pasivos y, según el entorno, en estáticos o dinámicos [17].

Otras clasificaciones atienden al tipo de mapa de entorno utilizado, mapas topológicos o mapas métricos, en métodos basados en *landmarks* o en métodos probabilísticos.

Cada uno de estos grupos no es excluyente, ya que pueden existir métodos de localización que pertenezcan a varios grupos. A continuación se describen los diferentes métodos de localización, haciendo especial hincapié en los métodos basados en *landmarks* y en los métodos probabilísticos.

- Métodos locales o globales

  Los métodos de localización locales son aquellos que requieren que la localización inicial del robot sea conocida, siendo sólo capaces de seguir la posición del robot. El objetivo de estas técnicas locales de localización es corregir el desfase en la posición del robot debido a los errores relativos a la odometría, consecuencia de los desplazamientos del robot.

  Por el contrario, los métodos de localización globales nos permiten determinar la posición del robot sin la necesidad de conocer ningún tipo de información referente a su posición inicial. Este tipo de técnicas presentan dos importantes ventajas frente a los métodos locales: no existe la necesidad de conocer la localización inicial del robot y, además, presentan mayor robustez, ya que es posible recuperar su posición a partir de posiciones falsas.

  A pesar de las ventajas de los métodos globales frente a los locales, son estos últimos los que se encuentran con mayor frecuencia, debido, en parte, a su menor complejidad.

- Métodos Activos y Pasivos

  La localización activa asume que, durante la localización, el algoritmo encargado de realizar este acometido, posee un control total o parcial sobre el robot, dando la oportunidad de incrementar la eficiencia y la robustez del proceso de localización.

En la localización pasiva el algoritmo de localización trabaja exclusivamente con los datos proporcionados por los sensores. El resto de consideraciones, como el movimiento del robot, no pueden ser controladas. La mayoría de los métodos son de naturaleza pasiva.

- Entornos estáticos y dinámicos

Esta clasificación atiende a la naturaleza del entorno. Nos referimos a entornos estáticos, como aquellos en los que toda la distribución del entorno es fija en el tiempo y lo único capaz de variar es la propia localización del robot.

Sin embargo, la mayoría de los entornos son dinámicos, ya que su apariencia o distribución puede cambiar de manera más o menos significativa. Por ejemplo, en una habitación, el mobiliario puede cambiar de lugar, las puertas o ventanas pueden estar abiertas o cerradas e, incluso, cabe la posibilidad de la existencia de otro robot móvil en el mismo entorno.

- Métodos Métricos y Topológicos

Cómo se explicó en el apartado 2.4., existen diferentes formas de representar el mapa del entorno que, a su vez, también supone un criterio de clasificación de los métodos de localización. Podemos distinguir los métodos métricos y los topológicos.

Los mapas métricos describen el entorno como un conjunto de objetos o de posiciones ocupadas en el espacio y los mapas topológicos, por medio de un grafo del entorno, representan la conectividad entre las diferentes regiones.

2.7. LOCALIZACIÓN BASADA EN MARCAS

Este método se basa en el reconocimiento de marcas o *landmarks*, que representan puntos característicos del entorno, siendo una referencia para determinar la localización del robot.

Para la consecución de este propósito es necesario que el robot móvil esté dotado de un sistema sensorial para recoger información del entorno, como pueden ser los sensores de visión artificial, ultrasonidos, infrarrojos, etc.

Una vez que estas *landmarks* son detectadas, se contrastan con la información que se tiene del entorno y, aplicando técnicas como la triangulación, entre otras, se determinará la posición del robot. Así, este

método de localización puede dividirse en 4 fases, tal y como se observa en el siguiente esquema (figura 2.7).

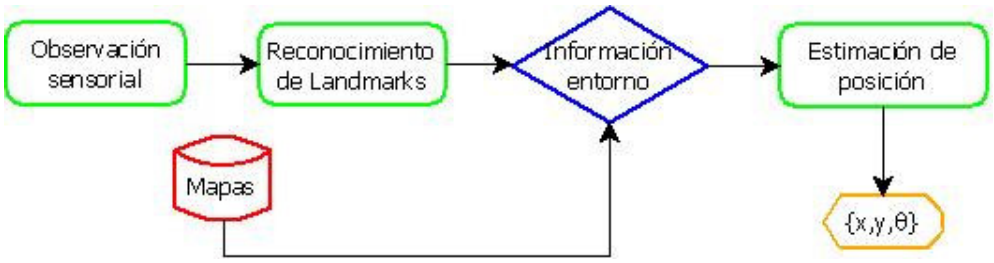
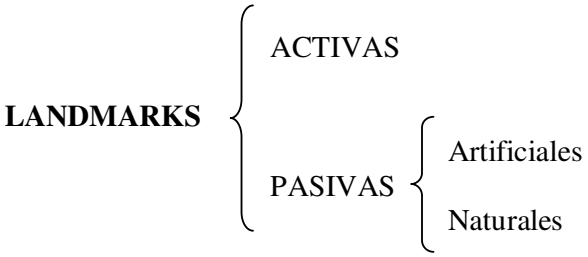


Figura 2.7.: Fases de localización de robots basada en *landmarks*

La localización por marcas se puede clasificar según la naturaleza de las marcas del entorno. Éstas pueden ser marcas activas o marcas pasivas. En estas últimas nos encontramos una subdivisión, pudiendo ser marcas artificiales o naturales.



2.7.1. MARCAS ACTIVAS

Las marcas activas son aquellas que emiten algún tipo de señal que informan sobre su localización. Un ejemplo de ellas son los satélites de los sistemas GPS, los faros ultrasónicos, los radiofaros y dipolos magnéticos, etc.

En la actualidad, el sistema más extendido y utilizado es el GPS, cuyas siglas en inglés significan "Global Positioning System". El Departamento de Defensa de los Estados Unidos se propuso crear un sistema de navegación de mayor precisión que desbancara a los sistemas presentes en la época, basados en el efecto Doppler, los cuales utilizaban la variación de frecuencia experimentada por las señales de radio transmitidas por los satélites, sistema TRANSIT.

El sistema GPS funciona mediante una red de 24 satélites en 6 órbitas sobre el globo terráqueo, a 20.200 Km, con trayectorias sincronizadas para cubrir toda la superficie de la Tierra. Cuando se desea determinar la posición, el receptor que se utiliza para ello localiza automáticamente, como mínimo, tres satélites de la red, de los que recibe unas señales indicando la identificación y la hora del reloj de cada uno de ellos. Con base en estas señales, el aparato sincroniza el reloj del GPS y calcula el tiempo que tardan en llegar las señales al equipo y, de tal modo, mide la distancia al satélite mediante triangulación (método de trilateración inversa), la cual se basa en determinar la distancia de cada satélite respecto al punto de medición. Conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los tres satélites. Conociendo además las coordenadas o posición de cada uno de ellos por la señal que emiten, se obtiene la posición absoluta o las coordenadas reales del punto de medición.

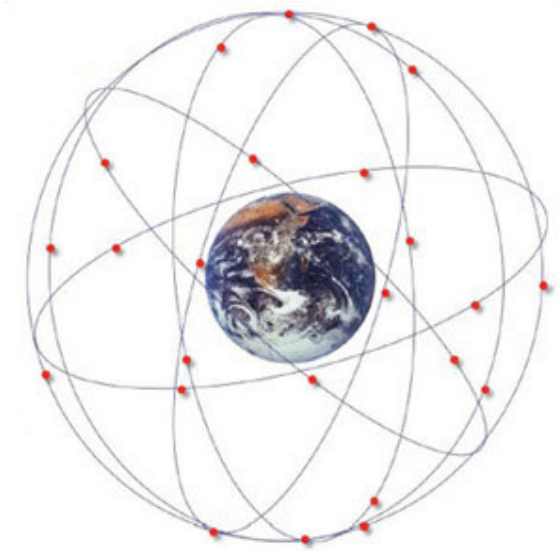


Figura 2.8.: Orbitas satélite GPS

La precisión del sistema GPS oscila entre los 10 y 50 metros. La precisión viene determinada por múltiples factores que pueden influir de forma negativa e introducir errores en la señal, entre los más destacados:

- Retraso de la señal en la ionosfera y la troposfera.
- Rebote de la señal en elementos urbanos (edificios) o elementos naturales (montañas).
- Errores orbitales donde los datos de la órbita no son completamente precisos.
- Número de satélites visibles.

- Geometría de los satélites visibles.
- Errores locales en el reloj del receptor GPS.

Uno de los factores más determinantes es el número de satélites disponibles o visibles, en caso de captar entre 7 y 9 satélites la precisión puede incrementar hasta valores de 2.5 m. Esta localización con este grado de precisión ha estado restringida tradicionalmente a aplicaciones militares.

En los últimos años, el desarrollo del GPS diferencial (DGPS), basado en la corrección de error utilizando una estación base de coordenadas conocidas, está permitiendo incrementar de forma importante la precisión, siendo cercana a 1 m.

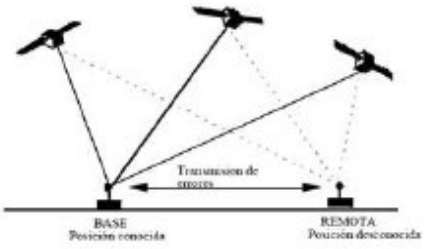


Figura 2.9.: Estación GPS diferencial

La principal desventaja de este tipo de *landmarks* es que la señal que emiten puede verse perturbada por las condiciones geográficas o, también, por las condiciones atmosféricas del entorno. Además de que la señal emitida por estas marcas no es de forma omnidireccional, por lo que el robot puede tener problemas para captarlas dependiendo del lugar donde se encuentre. Otra importante desventaja es que su costo de construcción y mantenimiento resulta excesivamente elevado.

2.7.2. MARCAS PASIVAS

Se basan en el reconocimiento de objetos perteneciente al entorno cercano del robot. Este tipo de marcas no emiten activamente ningún tipo de señal y, en consecuencia, el robot tiene que buscarlas activamente mediante sus sensores para poder ejecutar el proceso

posterior de localización. Es preciso referenciar estas marcas en un mapa global del terreno, *a priori* conocido, para poder determinar la posición absoluta del robot.

Como comentamos anteriormente, las marcas pasivas, atendiendo a su naturaleza, pueden ser artificiales o naturales:

- Marcas artificiales: Son colocadas intencionalmente en el entorno de trabajo, de tal forma que sean visibles a los sensores del robot [18]. Se emplean cuando éste desarrolla sus acciones en espacios de interior. Son múltiples los tipos de marcas artificiales existentes: códigos de barras, figuras geométricas, etc.

- Marcas naturales: Son aquellas que, de antemano, forman parte del entorno donde se mueve el robot. Representa un método más complejo que el caso anterior, ya que no sólo se tienen que detectar e identificar objetos naturales, sino que hay que decidir qué marcas se han de tener en cuenta. Para ello, se tienen que extraer características de los objetos, utilizando descriptores, que determinen la conveniencia de cada una de ellas. En ambientes interiores pueden ser las puertas, ventanas, lámparas, etc., mientras que para ambientes exteriores tenemos árboles, señales de tráfico, caminos, etc.
  La búsqueda de puntos característicos es más compleja en entornos exteriores que en espacios interiores.

La principal desventaja de estas *landmarks*, es que mientras más alejado se encuentre el robot de ellas, menos ajustada y precisa será la estimación del estado. También, al compararlas con las *landmarks* activas, se pone de manifiesto que ofrecen más dificultades a ser detectadas y requieren de mayor cantidad de proceso para poder identificarlas.

Dentro de los sistemas que utilizan marcas para localizarse, vamos a destacar el sistema más extendido en robótica, la localización por medio de balizas.

Las balizas estarán dispuestas en posiciones dentro de un entorno cerrado y, mediante el envío de señales, permitirán la localización al robot móvil. El número mínimo de balizas necesarias para obtener la posición del robot dependerá del sistema empleado; como es lógico, a mayor número de balizas, la precisión de la estimación obtenida también será mayor. Existen cuatro configuraciones posibles:

1. Se utilizan tres balizas colocadas en las lindes del terreno formando un triángulo. Las balizas emitirán señales independientes con el propósito de que sean captadas por el robot, el cuál mediante triangulación, teniendo en cuenta los ángulos entre las 3 balizas y el robot, calculará su posición en el entorno. Es una de las configuraciones más utilizadas.

2. En esta configuración sólo se hace uso de dos balizas. La posición del robot se obtiene utilizando los ángulos observados respecto al eje X.

3. De la misma forma que en el anterior caso, también se dispone de dos balizas con la diferencia de que la información utilizada son distancias en vez de ángulos. Con este método es necesario disponer de información adicional para poder determinar la localización del robot.

4. En esta última configuración sólo es necesario disponer de una baliza si se combina información angular y de distancia. Implica la necesidad de disponer de un sensor capaz de obtener ambas medidas.

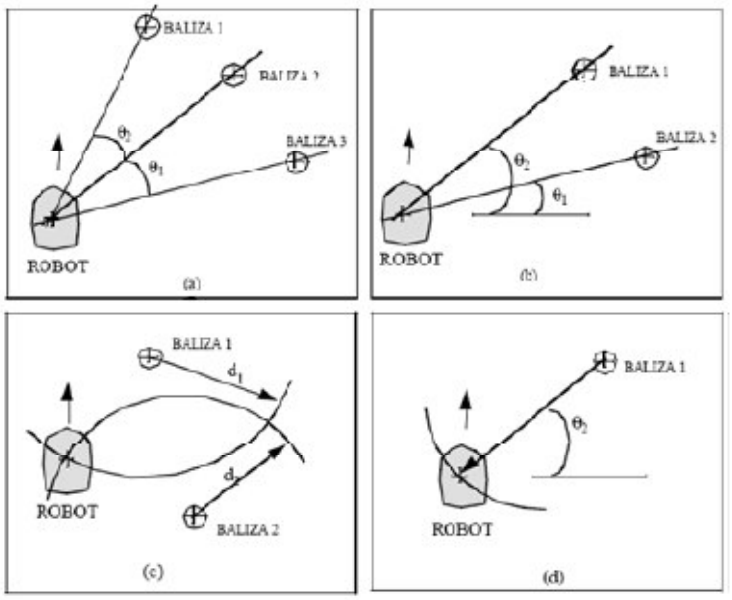


Figura 2.10.: Configuraciones de un sistema de balizamiento

Para todas las configuraciones es posible utilizar diferentes tipos de balizas, incluyendo las de naturaleza pasiva, y sensores. La precisión y fiabilidad que este tipo de estimación proporciona dependerá, en gran medida, de las características del sistema sensorial utilizado (infrarrojos, láser, ultrasonidos, etc.) y del número de balizas utilizadas en la triangulación.

2.7.3. TÉCNICAS DETERMINACIÓN DE LA POSICIÓN

Los métodos comúnmente utilizados para determinar la posición de un robot móvil a partir de la detección de marcas, activas y pasivas, son dos: La Trilateración, el cual se basa en las distancias desde el móvil a cada una de las *landmarks*, normalmente tres o más, y la Triangulación, el cual se basa en los ángulos respecto a las diferentes *landmarks* [19].

2.7.3.1. TRILATERACIÓN

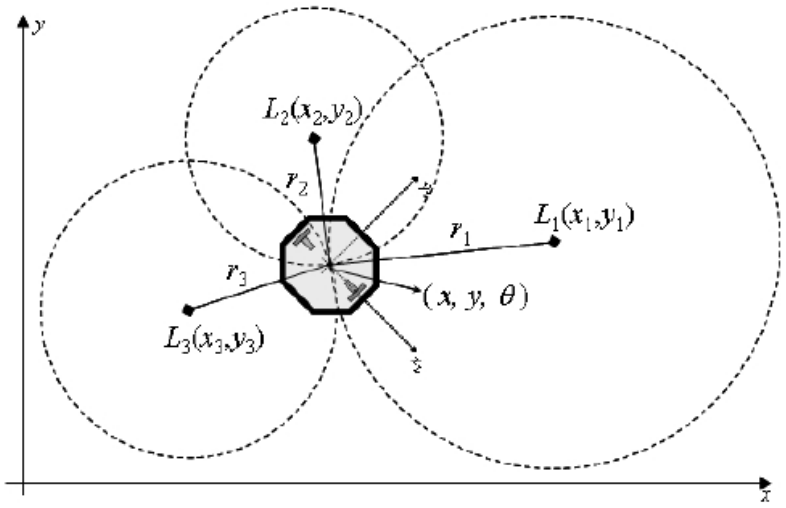


Figura 2.11.: Esquema de localización por trilateración. Este método se basa en la medición de las distancias $\{r_1, r_2, r_3\}$ del móvil a las *landmarks.*

La trilateración es la determinación de localización *(x,y)* de un robot móvil basado en la medición de las distancias (r_1,r_2,r_3) a 3 ó más *landmarks* cuyas coordenadas $\{(x_1,y_1)(x_2,y_2)(x_3,y_3)\}$ son conocidas, ver figura 2.11. En un plano 2D, la trilateración se puede definir como el problema de encontrar la intersección de tres circunferencias. Esto es encontrar la solución al sistema de ecuaciones cuadráticas expresadas en la ecuación (2.1).

$$\left.\begin{aligned}(x-x_1)^2+(y-y_1)^2&=r_1^2\\(x-x_2)^2+(y-y_2)^2&=r_2^2\\(x-x_3)^2+(y-y_3)^2&=r_3^2\end{aligned}\right\} \qquad (2.1)$$

Una forma simple de resolver el sistema dado (2.2), es sustrayendo la segunda y la tercera ecuación de la primera

ecuación. Así, el sistema se reduce a un sistema de primer orden con dos ecuaciones y dos incógnitas según,

$$\left.\begin{aligned} x_1^2 - x_2^2 - 2x\,(x_1 - x_2) + y_1^2 - y_2^2 - 2y\,(y_1 - y_2) &= r_1^2 - r_2^2 \\ x_1^2 - x_3^2 - 2x\,(x_1 - x_3) + y_1^2 - y_3^2 - 2y\,(y_1 - y_3) &= r_1^2 - r_3^2 \end{aligned}\right\} \quad (2.2)$$

La implementación práctica de la trilateración, usualmente, consiste en instalar 3 ó más marcas en lugares conocidos del entorno, e instalar un único receptor aborde del móvil. A pesar de la simplicidad y buena precisión que ofrece el método, en robótica móvil, este planteamiento de localización es insuficiente, ya que sólo se determina las coordenadas cartesianas del robot *(x,y)*, no obteniendo ninguna información referente a la orientación del robot respecto al sistema de coordenadas global.

2.7.3.2. TRIANGULACIÓN

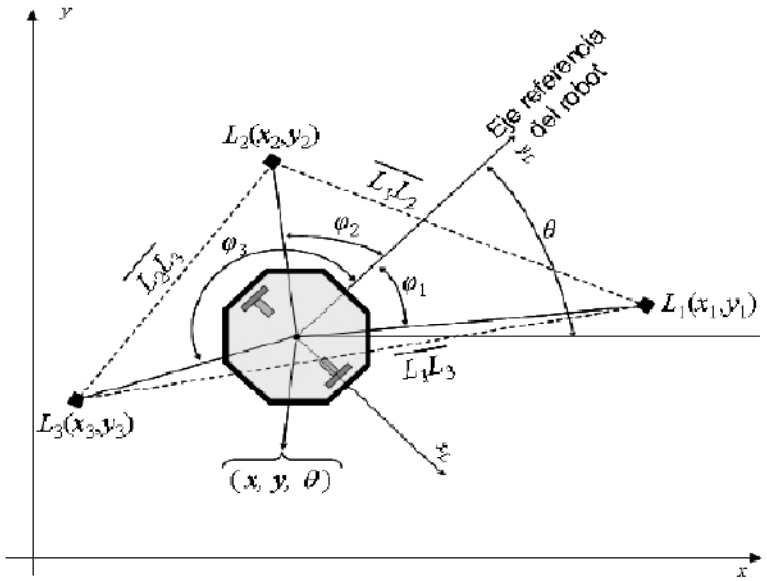


Figura 2.12.: Esquema de localización por triangulación. Este método se basa en la medición de los ángulos de vista $\{\Phi_1, \Phi_2, \Phi_3\}$ del móvil a las *landmarks.*

Por su parte, la triangulación es el proceso de determinar la localización del robot en el que, aparte de la posición *(x,y)*, también obtendríamos información acerca de la orientación *(θ)*. Para ello se hace uso de tres *landmarks* (L_1,L_2,L_3) cuyas respectivas localizaciones en el espacio cartesiano (x_i,y_i) son conocidas. En general, el proceso requiere que el robot tenga capacidad sensorial para detectar estas marcas y para medir la orientación o ángulo de vista hasta cada una de ellas $(\theta_1, \theta_2, \theta_3)$, respecto al sistema de referencia de él mismo [20], ver figura

2.12. Aplicando la ley de coseno, la distancia entre dos marcas cualesquiera se puede expresar como:

$$\left.\begin{array}{l} r_1^2 + r_2^2 - 2r_1 r_2 \cos\phi_{12} = \overline{L_1 L_2} \\ r_1^2 + r_3^2 - 2r_1 r_3 \cos\phi_{13} = \overline{L_1 L_3} \\ r_2^2 + r_3^2 - 2r_2 r_3 \cos\phi_{23} = \overline{L_2 L_3} \end{array}\right\} \quad (2.3)$$

Aquí, $\{r_1, r_2, r_3\}$ son las distancias desconocidas desde el robot hasta las *landmarks*, y $\{\overline{L_1L_2}, \overline{L_1L_3}, \overline{L_2L_3}\}$ son las distancias entre marcas que se consideran conocidas. Igualmente $(\theta_{12}, \theta_{13}, \theta_{23})$, surgen de sustraer los ángulos $(\theta_1, \theta_2, \theta_3)$, entre sí, por lo que también se consideran conocidos.

El sistema de ecuaciones de 2.3 puede resolverse utilizando un método de aproximación por mínimos cuadrados para obtener así las distancias desconocidas $\{r_1,r_2,r_3\}$. Una vez obtenidas estas distancias, se pasa a resolver la localización (x,y) del móvil usando el procedimiento descrito para las ecuaciones (2.1) y (2.2).

Una vez obtenida la localización del robot móvil, se puede computar el ángulo del segmento que desde el punto (x,y) hasta cualquiera de las marcas conocidas (x_i,y_i) y luego calcular el ángulo θ de orientación del móvil con la ecuación.

$$\theta = \angle^{(x,y)}_{(x_i,y_i)} - \phi_i \quad (2.4)$$

2.8. LOCALIZACIÓN BASADA EN MÉTODOS PROBABILÍSTICOS

En los métodos clásicos utilizados en robótica, el éxito de los algoritmos depende fundamentalmente de dos aspectos: de la precisión de los sensores del robot y de la exactitud de los modelos del entorno y del propio robot. Sin embargo, el cumplimiento de estas dos condiciones no es garantía de pleno éxito, debido a que los errores e incertidumbres siempre estarán presentes en el terreno real. A continuación, destacamos las principales adversidades:

- Ruido en los sensores. Las observaciones realizadas por los sensores normalmente ruidosas y la distribución estadística de este ruido no suele ser sencilla de modelar [21].

- Ruido en la detección de la posición. Los movimientos del robot no suelen ser exactos, ni tampoco detectados de forma precisa mediante odometría. Los errores de odometría son, además,

acumulativos. Pequeños errores en la rotación del robot puede tener efectos importantes en la estimación de los movimientos de traslación y en la determinación de su posición final [22].

- Entorno complejos y dinámicos. Los entornos interiores en los que se desplazan los robots suelen ser complejos y dinámicos, haciendo muy difícil mantener modelos consistentes de los mismos.

Ante esta problemática, los métodos o enfoques probabilísticos han demostrado que ofrecen un resultado más robusto que los métodos clásicos. Esto último, se le atribuye al hecho de que los métodos probabilísticos se apoyan en modelos que representan la información a través de funciones de probabilidad, lo que los hace más robustos de cara a las limitaciones del sensor y al ruido mismo en la cinemática del robot y en el modelo del entorno. Por otra parte, las limitaciones más citadas de los algoritmos probabilísticos son dos: la ineficiencia computacional, al tener que considerar todas las distribuciones de probabilidad del espacio de posiciones del robot; y la inherente necesidad de tener que aproximar discretamente la realidad continua del contexto del robot [23].

2.8.1. FILTRO DE KALMAN

El filtro de Kalman forma parte de la importante familia de estimadores de estado recursivo, comúnmente llamados filtros de Gauss. Históricamente, los filtros de Gauss constituyen las primeras implementaciones manejables del filtro de Bayes para espacios continuos. También son, con mucho, la familia más popular de las técnicas hasta la fecha, a pesar de presentar una serie de deficiencias. Las técnicas Gaussianas comparten la idea básica de que las creencias son representadas por distribuciones normales multivariantes [21]

Un filtro de Kalman [24, 25, 26, 27, 28] es un algoritmo de procesamiento de datos recursivo que calcula el estado de un sistema dinámico lineal incorporando los efectos del ruido proveniente tanto de las medidas como del modelo.

Cuando hablamos sobre el estado de un sistema, nos referimos a un vector x que consiste en m variables que describen algunas propiedades interesantes del sistema. Un ejemplo de un estado es la ubicación de un robot, que consiste en la coordenadas x e y, y la orientación de un robot θ. El hecho de que las variables del estado puedan estar afectadas por ruido y no sean directamente observables, hace que la estimación de estado sea difícil.

Un filtro de Kalman opera básicamente con un conjunto de ecuaciones matemáticas mediante las cuales se estima el estado de un

sistema lineal de forma eficiente, a través de una solución recursiva del método de mínimos cuadrados. Estas ecuaciones permiten implementar, de forma óptima, un estimador lineal del estado de un proceso, basándose en la información disponible en el instante anterior y corrigiendo este estado estimado a partir de la información disponible en el momento actual. Este estimador es óptimo en el sentido que minimiza el error cuadrático medio del estado estimado, el cuál suele verse afectado por ruido Gaussiano, según:

$$x_k = \mathbf{A}x_{k-1} \mid \mathbf{B}u_{k-1} \mid w_k \tag{2.5}$$

Esta ecuación define el modo en que cambia el estado *x*, el cual está relacionado en el tiempo con el estado anterior *(x_{k-1})* y con la entrada de control *u*, a través de la matriz de parámetros A y B, respectivamente.

La ecuación de salida del sistema, a la que se le ha incorporado el ruido de medida, se muestra a continuación:

$$z_k = \mathbf{H}x_k + v_k \tag{2.6}$$

Esta ecuación se conoce como el modelo de medida y, en ella, la lectura del sensor *z* está relacionada con el estado *x* a través de la matriz de parámetros H.

En las ecuaciones 2.5 y 2.6, las variables *w* y *v* representan el ruido de sistema y de medida respectivamente, los cuales se asume que no están correlacionados y que tienen distribución de probabilidad normal o Gaussiana, con media igual a cero.

$$\begin{cases} \mathrm{P}(w) \simeq \mathrm{N}(0, \mathbf{Q}) \\ \mathrm{P}(v) \simeq \mathrm{N}(0, \mathbf{R}) \end{cases} \tag{2.7}$$

Las ecuaciones matemáticas de Filtro de Kalman se basan en los subprocesos de predicción y corrección, los cuales son repetidos recursivamente. A partir de la estimación del valor precio de la variable fusionada *(x_{k-1})*, mediante la ecuación 2.5, el algoritmo pronostica el valor que tendrá la misma en el próximo instante ($X_k^{\text{pronosticada}}$). Luego se realiza una medición sensorial *(z_k)* que se utiliza para generar un término de corrección proporcional al error de predicción. Este término se añade a la variable pronosticada para generar, así, la variable corregida ($X_k^{\text{corregida}}$).

$$x_k^{\text{corregida}} = x_k^{\text{pronosticada}} + \mathbf{K}\left(z_k - \mathbf{H}x_k^{\text{pronosticada}}\right) \tag{2.8}$$

El factor de correción K, conocido como la ganancia de Kalman, se determina a partir de las matrices Q y R, las cuales representan la varianza/covarianza del ruido en el sistema y la del ruido en la medición.

$$\mathbf{K} = \frac{\mathbf{Q}}{\mathbf{Q} + \mathbf{R}} \tag{2.9}$$

De la ecuación anterior se observa que, si el ruido en el sistema es más importante que el ruido de la medición (Q>>R), entonces tiene más importancia la medición directa (K->1). Por el contrario, si el ruido en la medición es más importante que el ruido en el sistema (Q<<R), entonces se le da más importancia al estimado (K->0).

Tradicionalmente se ha utilizado este método para la fusión sensorial de las estimaciones provenientes de diferentes sensores. Todos los métodos existentes que utilizan este filtro son prácticamente parecidos en cuanto al modelo de movimiento del robot, y difieren en el modelo sensorial.

Este método presenta una alta precisión en la estimación de la posición del robot. Sin embargo, la consideración de una distribución gaussiana unimodal implica la necesidad de conocer la posición inicial del robot. Además, se debe asumir que el error asociado a las medidas se encuentra contaminado por ruido blanco o no correlacionado y que no existe relación de dependencia alguna entre las variables medidas.

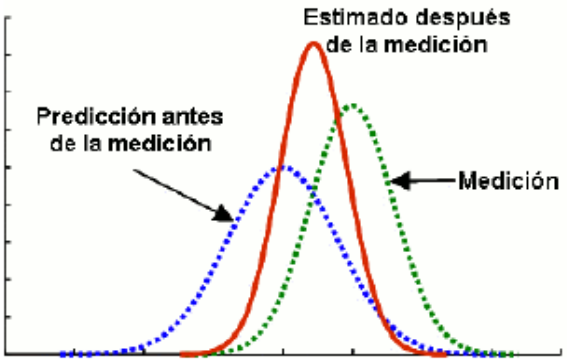


Figura 2.13.: Resultado típico de la fusión mediante el Filtro de Kalman

La aplicación de Filtro de Kalman en sistemas no lineales se hace mediante una extensión conocida como EKF (Extended Kalman Filter), la cual surge de la linealización de las ecuaciones de estado mediante la expansión en Series de Taylor de primer orden alrededor del estado estimado. Al aplicar esta operación de linealización, el Filtro de Kalman deja de ser óptimo, sin embargo, continua siendo *cuasi* óptimo para sistemas que presentan no-linealidades poco severas.

2.8.2. FILTRO DE PARTÍCULAS

La localización bayesiana se basa en técnicas probabilísticas y proporcionan una estimación de la posición de un robot móvil, a partir de las observaciones del robot y de las acciones de control realizadas para su movimiento. De esta forma, se pueden obtener los desplazamientos y, por tanto, la trayectoria seguida [21].

La regla de Bayes se utiliza para calcular la probabilidad *a posteriori* , que se puede expresar como la probabilidad $p(z_t|x_t)$ de la última medida z_t, ponderada por la función de probabilidad *a priori* $p(x_t/x_{t-1},a_{t-1})$ de la posición del robot, y la probabilidad *a priori* $p(x_{t-1})$:

$$p(x_t) = \alpha p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid x_{t-1}, a_{t-1}) p(x_{t-1}) \qquad (2.10)$$

donde α es un factor de normalización: $\int_{x_t} p(x_t) = 1$.

Los filtros de partículas [29] permiten resolver el problema de la localización bayesiana representando la función de probabilidad *a posteriori* que estima las posiciones más probables del robot. Los filtros de partículas presentan ventajas cuando la distribución de probabilidad es multimodal [21].

Los mencionados filtros resuelven el problema discretizando el espacio de estados (todas las posibles posiciones del robot en el espacio). Esta discretización se lleva a cabo definiendo celdas en todo el espacio en el que es posible que se encuentre el robot y asociando partículas a dichas celdas. La probabilidad de que el robot esté situado en cada una de las celdas se obtiene utilizando el modelo bayesiano, calculándose la probabilidad de cada partícula [29] y despreciándose aquellas cuyas probabilidad sea muy baja.

Algoritmo Básico

El filtro de partículas es una implementación no paramétrica del filtro bayesiano. Se aproxima la función de densidad o distribución de probabilidad *a posteriori* por un número finito de parámetros. La idea principal es representar la distribución de probabilidad *a posteriori, $p(x_t)$,* por un conjunto de muestras de estados aleatorios extraídas de esta distribución de probabilidad.

Sea *x* el estado del proceso, entonces la expresión (2.11) representa las ecuaciones del espacio de estado del proceso:

$$\begin{aligned} x_{t+1} &= f(x_t, v_t) + g(u_t, n_t) \\ z_t &= h(x_t, w_t) \end{aligned} \qquad (2.11)$$

Dónde z_t es la salida actual, u_t la entrada actual, w_k y n_k son muestras de una variable aleatoria, y *f, g, h* funciones conocidas.

Las muestras de una distribución de probabilidad *a posteriori* se denominan partículas, X_t. Cada partícula, *m*, representa un estado x_t^m en el instante *t*. Generalmente, el número de partículas es proporcional a $p(x_t)$.

El filtro de partículas computa $p(x_t)$ de forma recursiva a partir del estado anterior $p(x_{t-1})$; es decir, X_t dependerá de X_{t-1}.

El algoritmo básico del filtro de partículas es:

Filtro de partículas (X_{k-1}, u_k, z_k)

1. $\bar{X}_k = X_k = 0$
2. ***for*** $m = 1$ ***to*** M ***do***
3. $muestrear\ x_k^{[m]} \sim p\left(x_k \middle| u_k, x_{k-1}^{[m]}\right)$
4. $w_k^{[m]} = p\left(z_k \middle| x_t^{[m]}\right)$
5. $\bar{X}_k = \bar{X}_k + \left(x_k^{[m]} + w_k^{[m]}\right)$
6. ***for*** $m - 1$ ***to*** M ***do***
7. $extraer\ i\ con\ probabilidad\ \propto\ w_k^{[m]}$
8. $añadir\ x_k^{[i]}\ a\ X_t$
9. $return\ X_t$

Figura 2.14.: Algoritmo básico del filtro de partículas

Las entradas son el conjunto de partículas en el instante *t-1*, el control u_t y la medida z_t en el instante *t*.

El proceso se ejecuta, normalmente, sin tomar medidas. Cuando se toma una medida, el conjunto de partículas se puede modificar en ese instante haciendo un remuestreo.

El remuestreo es importante en la implementación del filtro de partículas porque reduce la varianza en la estimación del estado. La idea básica es que algunas muestras de la distribución de probabilidad son más significantes que otras. De esta forma, reajustando esta distribución a partir de las medidas cada cierto tiempo, la estimación del estado mejorará.

En la línea 4 del algoritmo presentado anteriormente se ha configurado el peso o importancia de cada partícula como la probabilidad de medir z_k (medida tomada) cuando el estado es x_k^m.

Estos pesos proporcionan la importancia de cada partícula. Así, el conjunto de partículas que tienen mayor peso serán las que mejor representen el estado. La elección de los pesos es clave para la precisión del algoritmo implementado. El filtro de partículas con el proceso de remuestreo puede describirse como sigue:

Filtro de partículas (X_{k-1}, u_k, z_k)

1. $\bar{X}_k = X_k = 0$
2. ***for*** $m = 1$ ***to*** M ***do***
3. $muestrear\ x_t^{[m]} \sim p\left(x_k \middle| u_k, x_{k-1}^{[m]}\right)$
4. $if\ existe\ z_k\ then$
5. $w_k^{[m]} = \eta p\left(z_k \middle| x_k^{[m]}\right)$
6. $\overline{X_k} = \overline{X_k} + \left(x_k^{[m]} + w_k^{[m]}\right)$
7. $N_{ef} \leftarrow 1 \Big/ \sum_{i=1}^{M} \left(w_k^{[i]}\right)^2$
8. ***if*** $N_{ef} < N_{\text{lim}}$ ***then***
9. ***for*** $m = 1$ ***to*** M ***do***
10. $extraer\ i\ con\ probabilidad \propto w_k^{[m]}$
11. $añadir\ x_k^{[i]}\ a\ X_t$
12. $return\ X_t$

Figura 2.15.: Algoritmo básico del filtro de partículas con proceso de remuestreo

En la línea 5, el coeficiente η es un factor de normalización tal que $\Sigma_{i=1}^{M} w_i = 1$.

El remuestreo tiene lugar en las líneas 9 y 10. La construcción del nuevo conjunto de partículas dependerá de los pesos de cada una de ellas, lo que repercutirá en una estimación del estado más precisa.

El remuestreo se realiza cuando el número efectivo de partículas N_{ef} es inferior a un cierto umbral N_{lim} (veáse línea 8 del algoritmo). Un N_{ef} cercano a 1 significa que hay una partícula con un peso mayor respecto al resto, y un N_{ef} cercano a M (número de partículas) significa que todas las partículas que representan el estado tienen el mismo peso.

3. COMPUTACIÓN EVOLUTIVA

La computación evolutiva es una rama de la inteligencia artificial que involucra problemas de optimización combinatoria. Se inspira en los mecanismos de la evolución biológica.

La aplicación de algoritmos evolutivos es un área de investigación relativamente reciente, pero hasta ahora se ha demostrado que estos algoritmos tienen ventajas significativas sobre otros métodos de optimización más tradicionales.

El hecho de que algunos científicos hayan elegido la evolución natural como una fuente de inspiración no es sorprendente. El poder de la evolución natural es evidente en la diversidad de las especies que conforman nuestro mundo, cada una de ellas adaptada a su ecosistema con el fin de sobrevivir.

3.1. ANTECEDENTES HISTÓRICOS

La teoría evolutiva universalmente aceptada hoy en día es el *Neodarwinismo*, basada en la teoría evolutiva clásica de Charles Darwin, la teoría selectiva de Weismann y las leyes de la genética de Mendel.

El *Neodarwinismo* establece que la diversidad actual de seres vivos es consecuencia de determinados procesos estocásticos que actúan en las especies y sus poblaciones. Estos procesos son: reproducción, mutación, competencia y selección.

La reproducción es una propiedad connatural de las especies. Toda especie que pierda su capacidad reproductiva está condenada a la extinción. La reproducción se logra a través de la transferencia de la información genética de los individuos a sus descendientes.

La mutación es una alteración de la información genética en el momento que ésta es transferida durante la reproducción.

La competencia es la consecuencia del crecimiento de las poblaciones de especies en un espacio con recursos finitos.

La selección es resultado de la competencia entre individuos y especies por ocupar el espacio y los recursos disponibles.

Así, la evolución surge como un resultado inevitable de la interacción de estos procesos físicos estocásticos básicos.

3.2. ALGORITMOS EVOLUTIVOS

Los algoritmos evolutivos son métodos de optimización y búsqueda de soluciones basados en los postulados de la evolución biológica. En ellos se mantiene un conjunto de entidades que representan posibles soluciones, las cuales se mezclan y compiten entre sí de tal manera, que las más aptas son capaces de prevalecer a lo largo del tiempo, evolucionando hacia mejores soluciones cada vez.

Siguiendo la terminología de la teoría de la evolución, las entidades que representan las soluciones al problema se denominan individuos o cromosomas, y el conjunto de éstos, población. Los individuos son modificados por operadores genéticos, principalmente el cruce, que consiste en la mezcla de la información de dos o más individuos; la mutación, que es un cambio aleatorio en los individuos; y la selección, consistente en la elección de los individuos que sobrevivirán y conformarán la siguiente generación. Dado que los individuos que representan las soluciones más adecuadas al problema tienen más posibilidades de sobrevivir, la población va mejorando gradualmente.

A continuación se muestra el pseudocódigo de un algoritmo evolutivo genérico.

- g:=0; Se genera la población inicial: P(0)
- Evaluación P(g)
- Hacer
    - P'(g):= Cruce [P(g)]
    - P" (g):= Mutación [P'(g)]
    - Evaluación[P"(g)]
    - P(g+1):= selección[P"(g)∪P(g)]
    - g:= g+1
- Mientras no se cumpla condición de término

Figura 3.1.: Esquema general de un algoritmo evolutivo

Los algoritmos evolutivos representan un método de búsqueda estocástico, que mantiene una población de individuos en cada generación *g*. Cada individuo representa una solución potencial del problema a estudio y se implementa como una cadena de bits o como números reales, dependiendo del método de codificación prefijado.

Donde:

- La inicialización es la creación de la población inicial *P(0)*, usualmente asignando valores aleatorios a cada individuo.
- *P(g)* representa una población de un número determinado de individuos en la generación g.
- La evaluación es la asignación de un indicador de aptitud (o capacidad para resolver el problema propuesto), para cada individuo de la población *P(g)*, mediante la aplicación de una función de salud.
- *P'(g)* es una población construida a partir de la aplicación del operador de cruce sobre la población *P(g)*.
- *P''(g)* es una población construida a partir de la aplicación del operador de mutación sobre la población *P'(g)*.
- La población de la generación siguiente *P(g+1)* se obtiene a partir de la selección de la unión de la población modificada *P''(g)* y *P(g)*, permaneciendo los de mejor salud.

- La condición de término es un criterio que indica cuándo se debe poner fin a la búsqueda. Este criterio puede ser un nivel de convergencia, un número de generaciones máximo, o un tiempo de ejecución máximo, entre otros.

Cabe hacer notar que los operadores de variación, la selección y la manera en que son utilizados dependen del enfoque que se esté usando.

Tras la inicialización, se aplica un conjunto de operadores de evolución y se evalúan los individuos de cada generación, permaneciendo los de mejor salud. Cuando se cumple la condición de término, se espera que el mejor individuo de la población represente una solución óptima.

Suele hablarse de tres paradigmas principales de algoritmos evolutivos:

- Programación evolutiva
- Estrategias evolutivas
- Algoritmos genéticos

Cada uno de estos paradigmas se originó independientemente y con distintas motivaciones. Actualmente, los algoritmos tienden a combinar características de estos tres y a incluir mecanismos de otros campos de estudio. Algunas de las tendencias actuales son las siguientes:

- Evolución diferencial
- Modelos probabilísticos
- Evolución simulada
- Algoritmos culturales
- Algoritmos meméticos
- Programación genética

En este capítulo, nos centraremos en explicar con detalle el paradigma de los algoritmos genéticos y, de igual forma, profundizaremos en lo referente a la rama de la evolución diferencial.

3.3. ALGORITMOS GENÉTICOS

Los algoritmos genéticos fueron propuestos en los años setenta por John H. Holland, inspirados en los mecanismos que se pueden observar en la evolución biológica. Actualmente, gozan de gran difusión gracias a su versatilidad para resolver un amplio rango de problemas.

Algunas de las características principales de los algoritmos genéticos son las siguientes [30]:

- No se trabaja directamente con los objetos, sino con la codificación de esos objetos, que pueden ser un número, un conjunto de parámetros, etc. A esta codificación le llamamos cromosoma y representa un vector de genes.

- Realizan la búsqueda mediante toda una generación de objetos, no buscan un único elemento.

- Utilizan una función objetivo que nos da la información del grado de adaptación de los diferentes individuos. Los cromosomas más aptos se reproducen con mayor frecuencia que los demás.

- La reproducción parte de la decodificación de los cromosomas de los progenitores, el cruce de los mismos y la codificación de los nuevos cromosomas formando descendientes.

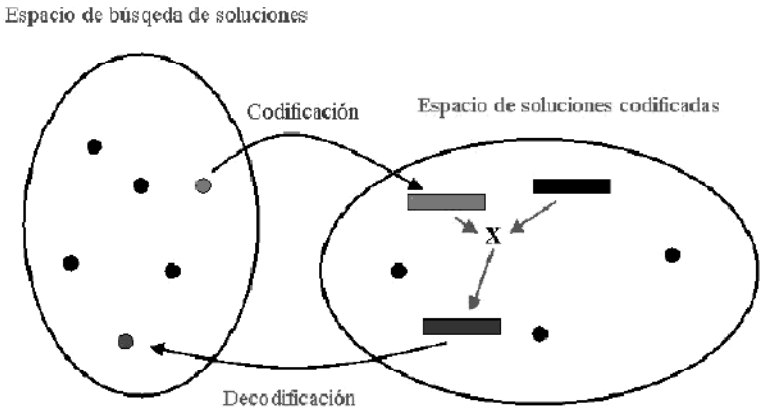


Figura 3.2.: Búsqueda genética

- Las reglas de transición son probabilísticas, no determinísticas.

- Las mejores características de los progenitores son heredadas por los descendientes a lo largo de la evolución de las poblaciones, consiguiendo que las generaciones futuras mejoren.

Basándose en estas características, Holland [31] creó un algoritmo que genera nuevas soluciones a partir de la unión de soluciones progenitoras utilizando operadores similares a los de la reproducción, denominados operadores genéticos.

Las soluciones de un cierto problema, o individuos, se codifican mediante cromosomas, formando una población. Los individuos que constituyen la población se esfuerzan por sobrevivir: una selección programada en el proceso evolutivo, inclinada hacia los individuos con mejores

características, determina aquellos individuos que formarán parte de la siguiente generación. El grado de adaptación de un individuo se evalúa de acuerdo al problema a resolver al aplicarle la función objetivo del problema (función de salud), obteniéndose un valor que se denomina salud o *"fitness"*. Los individuos se emparejan a través de ciertos mecanismos de reproducción, obteniéndose nuevas soluciones que se incorporan a la población, desplazando a las de menor salud. Las soluciones con mayor salud se emparejan con mayor frecuencia.

El algoritmo finaliza cuando la población converge hacia una solución considerada óptima, cuando no se mejora la salud media, o bien, en un cierto número de iteraciones.

La característica distintiva de los algoritmos genéticos respecto a las otras técnicas evolutivas consiste en su uso fundamental del cruce como operador principal, mientras que la mutación se utiliza como operador secundario tan sólo para agregar una nueva fuente de diversidad en el mecanismo de exploración del espacio de soluciones del problema. Inclusive la mutación puede llegar a ser un operador opcional o estar ausente en algunas variantes de algoritmos genéticos que utilizan otros operadores para introducir diversidad.

Los principales inconvenientes son [30]:

- El número de operaciones y, por tanto, el coste computacional podría llegar a ser muy alto si no se toman precauciones.

- Hasta el momento, no existe ninguna teoría matemática que nos asegure la convergencia.

3.3.1. ALGORITMO GENÉTICO BÁSICO

Un algoritmo genético puede presentar diversas variaciones, dependiendo de cómo se aplican los operadores genéticos (cruce, mutación), de cómo se realiza la selección y de cómo se decide el reemplazo de los individuos para formar la nueva población. En general, el pseudocódigo consiste en los siguientes pasos:

- **Inicialización**: Se genera aleatoriamente la población inicial, que está constituida por un conjunto de cromosomas, los cuales representan las posibles soluciones del problema. En caso de no hacerlo aleatoriamente, es importante garantizar que, dentro de la población inicial, se tenga la diversidad estructural de estas soluciones para tener una representación de la mayor parte de la población posible o, al menos, evitar la convergencia prematura.

- **Evaluación**: A cada uno de los cromosomas de esta población se aplicará la función de salud o función objetivo para evaluar la calidad que posee.

- **Selección:** Después de saber la aptitud de cada cromosoma, se procede a elegir los cromosomas que serán cruzados en la siguiente generación. Los cromosomas con mejor aptitud tienen mayor probabilidad de ser seleccionados.

- **Operadores genéticos:** Se emplean para determinar cómo será la nueva generación.

    - **Cruce:** El cruce es el principal operador genético. Representa la reproducción sexual, operando sobre dos cromosomas a la vez para generar dos descendientes donde se combinan las características de ambos cromosomas padres.

    - **Mutación:** Modifica al azar parte del cromosoma de los individuos, y permite alcanzar zonas del espacio de búsqueda que no estaban cubiertas por los individuos de la población actual.

    - **Reemplazo:** Una vez aplicados los operadores genéticos, se seleccionan los mejores individuos para conformar la población de la generación siguiente.

- **Condición de término:** El algoritmo se repetirá hasta converger hacia una solución óptima, que alcance un número máximo de interacciones o cuando no se registren cambios en la población.

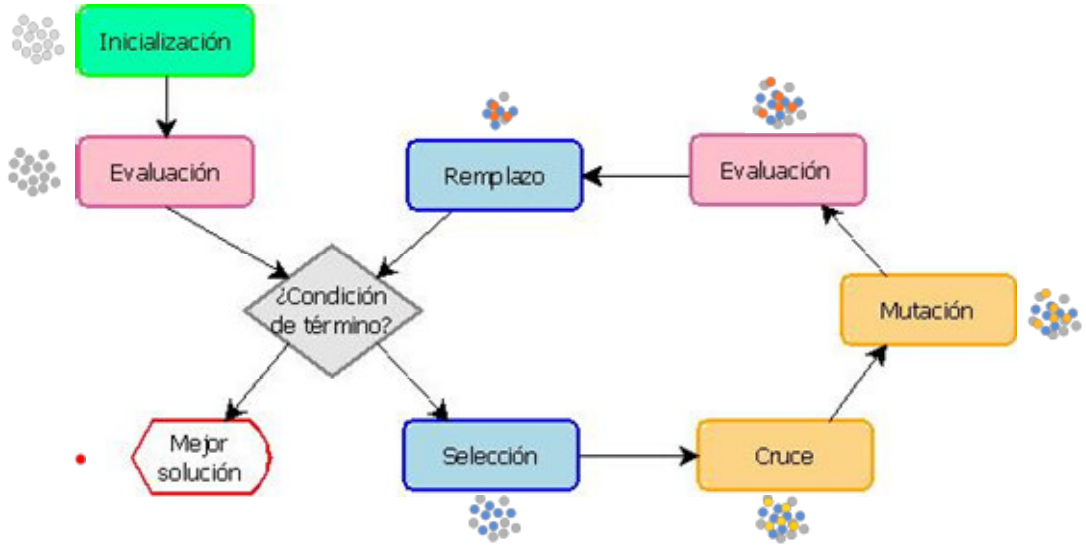


Figura 3.3.: Esquema del algoritmo genético

3.4. MÓDULOS DE LOS ALGORITMOS GENÉTICOS

L. Davis definió unos módulos comunes a todos los algoritmos genéticos, con unas técnicas asociadas a dichos módulos.

3.4.1. MÓDULO DE EVALUACIÓN

El módulo de evaluación se encarga de medir la salud de cada cromosoma a partir de la función a evaluar. Mientras mayor sea el valor de la función de coste, mejor es la solución.

3.4.2. MÓDULO DE POBLACIÓN

Este módulo recoge las técnicas utilizadas para mantener y manipular los cromosomas de una misma generación. Algunas de ellas son:

- <u>TÉCNICAS DE REPRESENTACIÓN</u>

Dado que un algoritmo genético trabaja sobre cromosomas, vector de genes, se debe definir una función de codificación sobre los puntos del espacio de soluciones.

Habitualmente, los algoritmos genéticos utilizan codificaciones binarias. Los cromosomas se codifican como una cadena de bits representada en valores binarios (ceros y unos). Un cromosoma es un vector de genes, mientras que el valor asignado a un gen se denomina alelo. Cada bit de la cadena puede representar alguna característica de la solución y la cadena completa puede representar una solución potencial del problema.

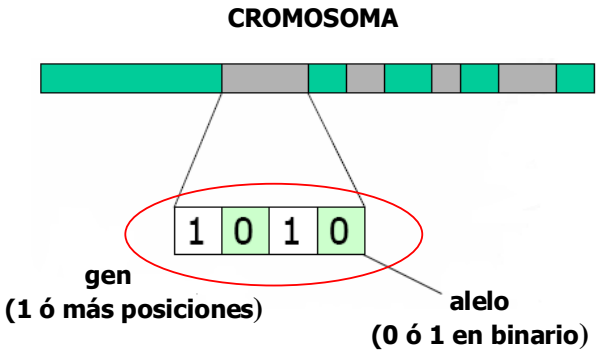


Figura 3.4.: Representación estructural de un cromosoma

Otras técnicas de codificación son utilizadas con menor frecuencia, como puede ser la codificación basada en números reales, donde cada bit es un número real. En particular, este método se utiliza en problemas con restricciones. Otra técnica es la codificación basada en ratios, donde la suma de los alelos debe ser constante; o la codificación basada en el orden donde existe una relación entre un gen actual y el que le precede.

- TÉCNICAS DE INICIALIZACIÓN

Es la creación de la primera generación de cromosomas; habitualmente se realiza de forma aleatoria.

- TÉCNICAS DE BORRADO

Se utilizan para concretar los cromosomas progenitores que desaparecen de la generación a favor de los descendientes; normalmente se elimina a los cromosomas que presentan menor salud.

- TÉCNICAS DE EVALUACIÓN DE LA SALUD

Son las técnicas que establecen el valor de la salud a partir de la evaluación de la función objetivo. En biología, a esta función se le denomina función de salud o *fitness*. Todo cromosoma tiene un valor asociado de *fitness* que evalúa la aptitud del individuo para resolver el problema en cuestión. La influencia de esta función dentro del algoritmo genético es importante, ya que guía al mecanismo de exploración a actuar representado el entorno que evalúa la salud de un individuo solución para la resolución del problema.

- TÉCNICAS DE SELECCIÓN

Estas técnicas serán las encargadas de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no.

Puesto que se trata de imitar lo que ocurre en la naturaleza basándonos en la teoría de la evolución de Darwin en la que "los mejores deben sobrevivir", se ha de otorgar un mayor número de oportunidades de reproducción a los individuos más aptos. Por lo tanto, la selección de un individuo estará relacionada con su función de salud. No se debe, sin embargo, eliminar por completo las opciones de reproducción de los individuos menos aptos, pues en pocas generaciones la población se volvería homogénea.

A continuación se detallan algunos de los métodos de selección:

- Método de la Ruleta: Propuesto por De Jong, es el método de selección más utilizado [32].

  Se supone una población de puntos codificados en cadenas binarias (se limita el tamaño de la población a cuatro cromosomas) [30].

  Como se puede observar en la tabla 3.1, cada cromosoma tiene asociado un valor de salud de acuerdo a una determinada función salud. Por otra parte, se define la probabilidad de selección de cada individuo como el cociente entre su salud y la suma de la salud de todos los individuos de manera que, aquéllos individuos con mayor salud, tengan mayor porcentaje de selección. El número de veces esperado que cada cadena sea seleccionada para reproducirse, no es más que su probabilidad por el tamaño de la población (así el tamaño de la población generada será igualmente de cuatro muestras).

Cromosoma	Salud asociada	Probabilidad selección	Nº copias esperadas
1	25	0.087	0.35
2	81	0.283	1.13
3	36	0.125	0.51
4	144	0.503	2.01
TOTAL	286	1.00	4.00

Tabla 3.1.: Cromosomas de una población y su salud asociada al método de selección de ruleta

El proceso de selección es el siguiente: se supone una ruleta dividida en cuatro porciones proporcionales a la probabilidad de cada individuo, donde al individuo de mayor salud (mayor probabilidad) se le asigna la mayor porción. Si se elige un punto de la ruleta de forma aleatoria, éste estará comprendido dentro de una de las porciones y el individuo asociado a la misma será seleccionado para la reproducción. De esta forma, los individuos de mayor salud (mayor porción ruleta) tienen una mayor probabilidad de ser seleccionados, mientras que los de poca salud (menor porción ruleta) corren una suerte contraria.

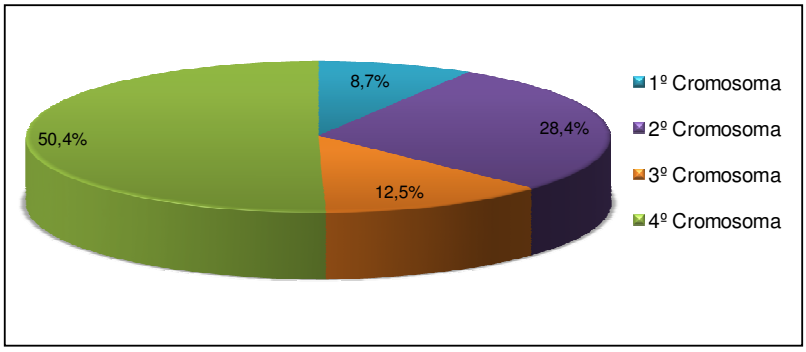


Figura 3.5.: Método de la ruleta

El número de veces que debemos de repetir el proceso de la ruleta es igual al número de la población que se desee generar, dado que por cada pareja de padres seleccionados, se obtienen en la reproducción una pareja de hijos.

- Método por Torneo: La idea principal de este método consiste en realizar la selección en base a comparaciones directas entre individuos. Existen dos versiones:

  - Determinística
  - Probabilística

  En la versión determinística se selecciona al azar un número *m* de individuos, normalmente se escoge m=2. De entre los individuos seleccionados, se comparan entre sí y se selecciona el más apto para que forme parte de la siguiente generación. Una característica de esta versión, es que el peor cromosoma de la población nunca será seleccionado como padre.

  La versión probabilística únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor cromosoma, se genera un número aleatorio entre cero y uno, parámetro p, que permanece constante para todo el proceso evolutivo. Se escogerá el individuo más apto si también es mayor que el parámetro p; en caso contrario, se seleccionará el menos apto. Generalmente p toma valores en el rango de $0,5 < p \leq 1$.

  Después de cada selección, los cromosomas son devueltos a la población y otro torneo comienza.

- Selección de Estado Uniforme: La idea principal de esta selección es que gran parte de los cromosomas deben sobrevivir a la siguiente generación.

  En cada generación se selecciona un número determinado de cromosomas que poseen las mejores aptitudes para la creación de nuevos descendientes. A continuación, la parte de cromosomas que presenta aptitud baja en la población inicial se eliminan y se sustituyen por los descendientes recientemente creados. El resto de la población sobrevive a la nueva generación.

3.4.3. MÓDULO DE REPRODUCCIÓN

El módulo de reproducción engloba las técnicas utilizadas para la creación de nuevos cromosomas a partir de los progenitores. A continuación se detallan los principales operadores genéticos, que son los encargados de determinar las diferentes técnicas reproductivas.

3.4.3.1. CRUCE

Una vez seleccionados los individuos, éstos son recombinados para producir la descendencia que se insertará en la siguiente generación. Tal y como se ha indicado anteriormente, el cruce es una estrategia de reproducción sexual.

Su importancia para la transición entre generaciones es elevada, puesto que las tasas de cruce con las que se suele trabajar rondan el 90 %.

La idea principal del cruce se basa en que se toman dos individuos correctamente adaptados al medio y se obtiene una descendencia que comparta genes de ambos. Al compartir las características buenas de dos individuos, la descendencia o, al menos, parte de ella, deberá tener una salud mayor que cada uno de los padres por separado. Si el cruce no agrupa las mejores características en uno de los hijos y la descendencia tiene una peor salud que la de los padres no significa que se esté dando un paso atrás, ya que en cruces posteriores se puede recuperar la salud perdida.

Existen multitud de técnicas de cruce. Sin embargo, los más empleados son los que se detallarán a continuación.

3.4.3.1.1. Cruce de un punto

Es la más sencilla de las técnicas de cruce. Una vez seleccionados dos individuos, se cortan sus cromosomas por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola. Se intercambian las colas entre los dos individuos para generar los nuevos descendientes. De esta manera, ambos descendientes heredan información genética de los padres.

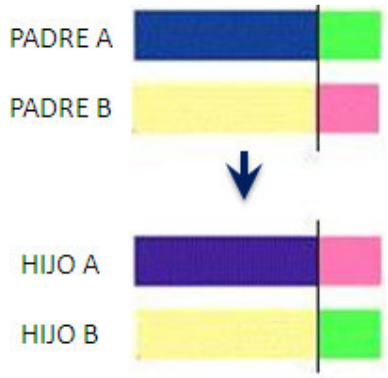


Figura 3.6.: Técnica cruce de un punto

3.4.3.1.2. Cruce de dos puntos

Se trata de una generalización del cruce de 1 punto. En vez de cortar por un único punto los cromosomas de los padres como en el caso anterior, se realizan dos cortes. Deberá tenerse en cuenta que ninguno de estos puntos de corte coincida con el extremo de los cromosomas para garantizar que se originen tres segmentos. Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre.

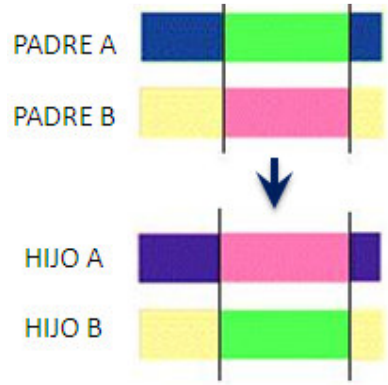


Figura 3.7.: Técnica cruce de dos puntos

Generalizando, se pueden añadir más puntos de cruce dando lugar a algoritmos de cruce multipunto. Sin embargo, existen estudios que desaprueban esta técnica [33]. Aunque se admite que el cruce de 2 puntos aporta una sustancial mejora con respecto al cruce de un sólo punto, el hecho de añadir un mayor número de puntos de cruce reduce el rendimiento del algoritmo genético. El problema principal de añadir nuevos puntos de cruce radica en que es más fácil que los segmentos originados sean corrompibles, es decir, que por separado quizás pierdan las características de salud que posean conjuntamente. Sin embargo, no todo son desventajas y añadiendo más puntos de cruce se consigue que el espacio de búsqueda del problema sea explorado más a fondo.

3.4.3.1.3. Cruce Uniforme

El cruce uniforme es una técnica completamente diferente de las vistas hasta el momento. Cada gen de la descendencia tiene las mismas probabilidades de pertenecer a uno u otro padre.

Aunque se puede implementar de muy diversas formas, la técnica implica la generación de una máscara de cruce con valores binarios. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre. Si, por el contrario, hay un 0, el gen se copia del segundo padre. Para producir el segundo descendiente, se intercambian los papeles de los padres, o bien, se intercambia la interpretación de los unos y los ceros de la máscara de cruce.

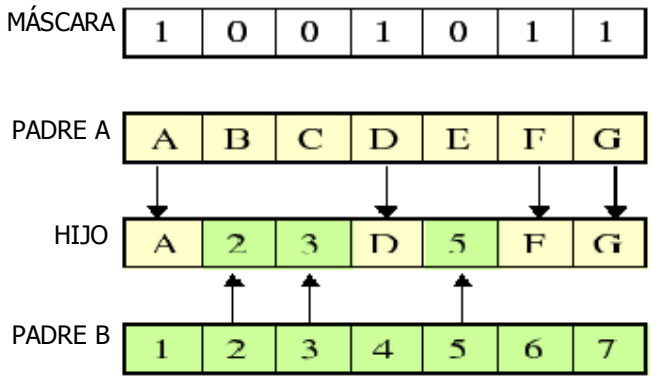


Figura 3.8.: Cruce Uniforme

3.4.3.2. MUTACIÓN

La mutación de un individuo provoca que alguno de sus genes, generalmente uno sólo, varíe su valor de forma aleatoria.

Aunque se pueden seleccionar los individuos directamente de la población actual y mutarlos antes de introducirlos en la nueva población, la mutación se suele utilizar de manera conjunta con el operador de cruce. Primeramente se seleccionan dos individuos de la población para realizar el cruce. Si el cruce tiene éxito, entonces uno de los descendientes, o ambos, se muta con cierta probabilidad. Se imita, de esta manera, el comportamiento que se da en la naturaleza, pues cuando se genera la descendencia siempre se produce algún tipo de error, por lo general, sin mayor trascendencia, en el paso de la carga genética de padres a hijos.

La probabilidad de mutación es muy baja, generalmente menor al 1 %. Esto se debe, sobre todo, a que los individuos suelen tener una salud menor después de mutados. Sin embargo, se realizan mutaciones para garantizar que ningún punto del espacio de búsqueda tenga una probabilidad nula de ser examinado.

Tal y como se ha comentado, la mutación más usual es el reemplazo aleatorio. Éste consiste en variar aleatoriamente un gen de un cromosoma. Si se trabaja con codificaciones binarias consiste simplemente en negar un bit. También es posible realizar la mutación intercambiando los valores de dos alelos del cromosoma. Con otro tipo de codificaciones no binarias existen otras opciones, como incrementar/decrementar a un gen una pequeña cantidad generada aleatoriamente o multiplicar un gen por un valor aleatorio próximo a 1.

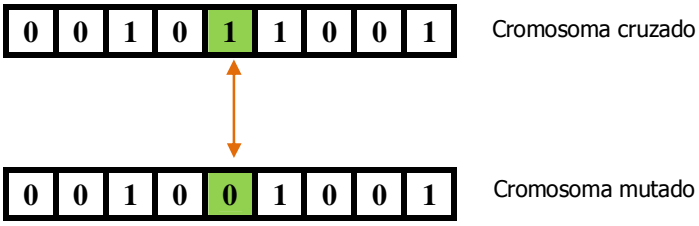


Figura 3.9.: Técnica de mutación

3.4.3.3 ELITISMO

El elitismo es otra estrategia reproductiva para la obtención de una nueva generación a partir de la anterior. A diferencia del cruce, se trata de una estrategia de reproducción asexual. Consiste, simplemente, en la copia de un individuo en la nueva generación.

Cuando se crea una nueva población mediante cruce o mutación se tiene una alta probabilidad de que se pierda el mejor cromosoma. Este método fuerza al algoritmo genético a retener algún número de los mejores individuos de cada generación.

El porcentaje de copias de una generación a la siguiente es relativamente reducido pues, en caso contrario, se corre el riesgo de una convergencia prematura de la población hacia ese individuo. De esta manera, el tamaño efectivo de la población se reduciría notablemente y la búsqueda en el espacio del problema se focalizaría en el entorno de ese individuo.

Lo que, generalmente, se suele hacer es seleccionar dos individuos para el cruce y, si éste finalmente no tiene lugar, se insertan en la siguiente generación los individuos seleccionados.

3.5. EVOLUCIÓN DIFERENCIAL

La Evolución Diferencial (ED) es una técnica estocástica de búsqueda directa, la cual ha mostrado ser un método robusto de búsqueda en una amplia gama de problemas [34].

En 1995, Rainer Storn y Kenneth Price, propusieron este nuevo método de algoritmo evolutivo. Originalmente, el método estaba enfocado en la resolución del problema de ajuste de polinomio de Tchebychev, utilizando una variante del método llamado Recocido Genético (Genetic Annealing), el cual había sido desarrollado por Price el año anterior.

La ED surgió como una nueva heurística para la minimización de funciones no lineales y no diferenciables en el espacio continuo. Storn y Price [35] demostraron que este método obtenía mejores resultados y resultaba ser más rápido que otros métodos conocidos hasta el momento.

Actualmente, la Evolución Diferencial representa una de las principales corrientes de investigación de la computación evolutiva. Entre las ventajas fundamentales, se puede destacar el hecho de que requieren menos parámetros de ajuste, son robustos y considerablemente más eficaces para resolver problemas de optimización que la mayoría de los algoritmos evolutivos.

La ED posee una amplia gama de aplicaciones, de entre la cuales se encuentran [36]: diseño de filtros digitales, entrenamiento de redes neuronales, diseño óptimo de tubos para intercambio de calor, optimización de procesos químicos no lineales, optimización de tuberías de agua, diseño de redes de transmisión de gas, calendarización óptima de satélites, etc.

3.5.1. DESCRIPCIÓN DEL ALGORITMO

Al igual que otros algoritmos de esta categoría, la evolución diferencial mantiene una población de soluciones candidatas, las cuales se cruzan y mutan para producir nuevos individuos, los cuales serán elegidos de acuerdo al valor de su función de salud. Lo que caracteriza a la ED es el uso de vectores de prueba, los cuales compiten con los individuos de la población actual a fin de sobrevivir.

El algoritmo asume que las variables del problema a optimizar están codificadas como un vector de números reales. La longitud de estos vectores es igual al número de variables del problema.

La Evolución Diferencial se compone básicamente de 4 pasos:

- Inicialización
- Mutación
- Cruce
- Selección

3.5.1.1. INICIALIZACIÓN

Como primer paso del algoritmo, se ejecuta la inicialización, se generan np individuos aleatoriamente que conforman a la población inicial $P_0=\{ _1, _2,\ldots, _{np}\}$. En la mayoría de los casos, los individuos están uniformemente distribuidos en el espacio de búsqueda dentro de los límites definidos por los valores mínimos y máximos de cada variable, tal y como se muestra en la ecuación (3.1).

$$x_{i,j}=x_j^{min}+U(0,1)\left(x_j^{max}-x_j^{min}\right)\ \forall i\in\{1,\ldots,np\},\forall j\in\{1,\ldots,np\} \quad (3.1)$$

$X_{ij,}$ *i* representa el índice del individuo en la población (i=1,…,np), *j* representa el índice de la variable en el individuo (j=1,…,n), U(0,1) representa una variable aleatoria uniformemente distribuida en el rango de [0,1] y x_j^{max} x_j^{min} representan los límites superiores e inferiores, respectivamente, que restringen el dominio de las variables del problema.

Después de la inicialización, la población es sometida a un proceso iterativo de mutación, cruce y selección, que se realizará repetidas veces hasta que una condición de término sea satisfecha (número de generaciones, tiempo transcurrido o calidad de solución alcanzada, entre otras).

3.5.1.2. MUTACIÓN

Básicamente se generan nuevos individuos añadiendo, a la diferencia entre vectores de población (tras multiplicar por un cierto peso F), un tercer vector.

La mutación puede realizarse de muchas maneras diferentes, para presentar cómo funciona la mutación en el caso de algoritmos *differential evolution* se muestra a continuación una estrategia de mutación descrita en [37] e implementada con éxito, ecuación (3.2):

- ✓ DE/RAND/1: Evolución diferencial. Orden 1. Centrada en un vector de la población elegido aleatoriamente.

$$\vec{u}_i = \vec{x}_{r_1} + F(\vec{x}_{r_2} - \vec{x}_{r_3}) \quad (3.2)$$

Donde $r_1, r_2, r_3 \in \{1,..,np\}$ son seleccionados aleatoriamente, además $r_1 \neq r_2 \neq r_3 \neq i$. $F \in [0,2]$ es el parámetro que controla la tasa de mutación.

3.5.1.3. CRUCE

Para aumentar la diversidad de los nuevos vectores de parámetros se introduce el cruce. Con la aplicación de los operadores de mutación y cruce se persigue generar un sólo descendiente o vector candidato $\vec{U}_i$ por cada vector padre $\vec{X}_i$ que pertenece a la población.

Existen dos operadores de cruce, que se muestran a continuación.

- Operador de cruce binomial:

$$u_{i,\forall j \in \{1,\dots,np\}} = \begin{cases} x_{r3,j} + F\left(x_{r1,j} - x_{r2,j}\right) & si\ U(0,1) < CR\ o\ j = j_{rad} \\ x_{i,j} & en\ caso\ contrario \end{cases} \quad (3.3)$$

- Operador de cruce exponencial:

$$u_{i,\forall j \in \{1,\dots,np\}} = \begin{cases} x_{i,j} & si\ U(0,1) > CR\ o\ j \neq j_{rad} \\ x_{r3,j} + F\left(x_{r1,j} - x_{r2,j}\right) & en\ caso\ contrario \end{cases} \quad (3.4)$$

En la siguiente figura se puede ver esquemáticamente el ejemplo de estos dos operadores de cruce.

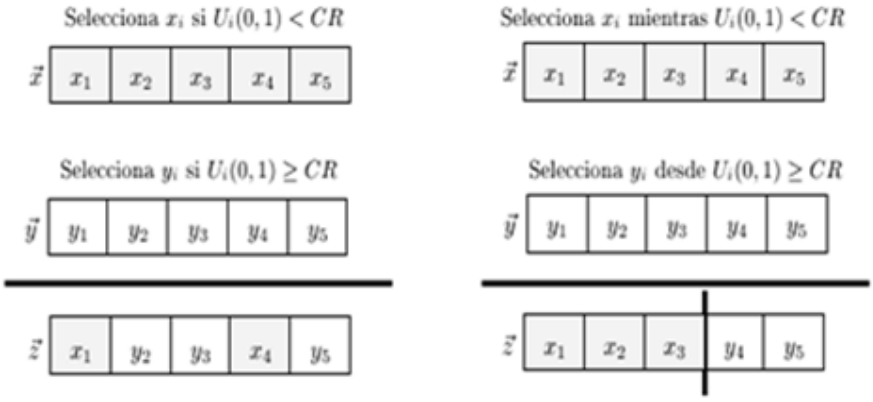


Figura 3.10.: Izquierda: Cruce binomial. Derecha: Cruce exponencial

3.5.1.3. SELECCIÓN

Tras realizar la mutación y el cruce hay que determinar qué elementos formarán la nueva generación

Para llevar a cabo esta selección, se compara la salud del vector resultante $\vec{U}_i$ con la salud del vector inicial $\vec{X}_i$, de manera que el vector de la generación siguiente será aquél que tenga el mejor valor de función de desempeño.

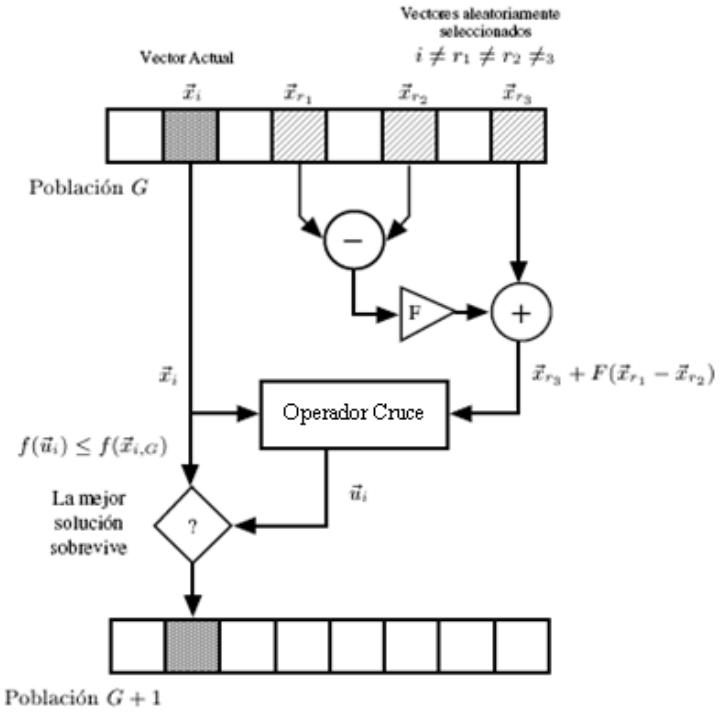


Figura 3.11.: Diagrama del algoritmo de Evolución Diferencial

Una vez determinados los individuos que conforman la siguiente población, el ciclo se repite hasta que el problema sea resuelto o todos los vectores objeto converjan a un punto o no se logre ninguna mejora después de varias iteraciones o después de ejecutarse un número fijo de generaciones.

En la figura 3.11, se puede observar de forma gráfica el funcionamiento del algoritmo de evolución diferencial.

3.5.2. AJUSTE DE PARÁMETROS

Los algoritmos de evolución diferencial se diferencian principalmente de los algoritmos genéticos en el orden en el que se realizan las operaciones de cruce y mutación.

3.5.2.1. TAMAÑO DE LA POBLACIÓN

Storn y Price sugieren que se comience el algoritmo con una población de tamaño alrededor de 10 veces el número de parámetros a optimizar. Si se obtiene buen resultado, se puede disminuir el tamaño de la población.

3.5.2.2. MUTACIÓN

Puesto que la mutación es la primera operación que se lleva a cabo, hay que determinar en primer lugar qué estrategia se utiliza.

En contraste con los algoritmos genéticos, los algoritmos de evolución diferencial utilizan una probabilidad de mutación pm =1; es decir, se mutan todos los individuos de la población.

Además hay que determinar el valor del factor de variación diferencial (factor de mutación). Normalmente, un algoritmo de evolución diferencial funciona bien para valores de $F \in [0.5, 1]$.

3.5.2.3. CRUCE

Los algoritmos de evolución diferencial son mucho más sensibles a la variación del factor de mutación F que a la variación de la constante de cruce CR.

Se podría decir que CR es un elemento de ajuste más fino. Valores altos de CR como CR=1 aumentan la velocidad de convergencia en caso de que ésta tenga lugar. Pero hay veces que el valor de CR debe disminuir, incluso hasta CR=0, para hacer que el algoritmo sea suficientemente robusto para un problema en particular.

Si se utiliza un cruce binomial, el valor de CR suele ser mayor que en el caso de utilizar un cruce de tipo exponencial. De todas formas, al igual que ocurre con el factor de mutación, un factor de cruce hace que el algoritmo funcione correctamente CR ϵ [0.5, 1] para la mayoría de los problemas.

La elección del método de cruce generalmente no es importante, aunque Price concluye que un cruce binomial nunca es peor que uno exponencial.

4. ROBOT MÓVIL Y TOOLBOX

En este capítulo nos centraremos en la descripción de dos bloques bien diferenciados del robot móvil en cuestión.

En primer lugar, trataremos la parte que comprende el *Hardware* del robot. Se detallarán el funcionamiento y las características tanto de los motores como de los sensores y se describirán los aspectos constructivos más relevantes.

En segundo lugar, focalizaremos nuestra atención en los diferentes *Software* existentes para el control del robot, dedicando especial atención al utilizado para la consecución del presente proyecto.

4.1. ROBOT MÓVIL

Lego Mindstorms es un equipo de robótica fabricado por la empresa Lego, el cual posee elementos básicos de las teorías robóticas, como la unión de piezas y la programación de acciones, en forma interactiva. Este robot fue comercializado por primera vez en septiembre de 1998 con el nombre de Robotics Invention System, bloque RCX en la figura 4.1.

En el presente proyecto, se utiliza la última versión del producto, Lego Mindstorms NXT, lanzada en enero de 2006. La diferencia más significativa respecto a su antecesora, es que esta versión utiliza el bloque NXT que representa una versión mejorada a partir del bloque RCX, considerado el predecesor y precursor de los bloques programables de Lego.

También cabe destacar las mejoras que incorporan los motores y sensores en sus prestaciones, así como las posibilidades de comunicación existentes.

Figura 4.1.: A la izquierda: Unidad de control RCX. A la derecha: Nueva generación, unidad de control NXT

Lego comercializa la generación NXT en dos versiones: Retail Version y Education Base Set. La principal ventaja de la versión Educacional, es que incluye una batería de Litio recargable y un mayor número de piezas y sensores, pero presenta el inconveniente de la necesidad de adquirir el *software* según el tipo de licencia: personal, educativo, etc.

4.2. COMPONENTES DEL ROBOT MÓVIL

4.2.1. UNIDAD DE CONTROL

El NXT es la unidad de control del robot MINDSTORMS. Es un ladrillo inteligente controlado por ordenador que permite al robot cobrar vida y realizar diferentes operaciones [40].

El ladrillo está compuesto por:

- Tres puertos de salida para conectar los motores: puerto A, B y C.
- Cuatro puertos de entrada para conectar los sensores: puerto 1, 2, 3 y 4.
- Un puerto de conexión USB.
- Interfaz para la conexión Bluetooth.
- Un altavoz, botones de control y una pantalla LCD.

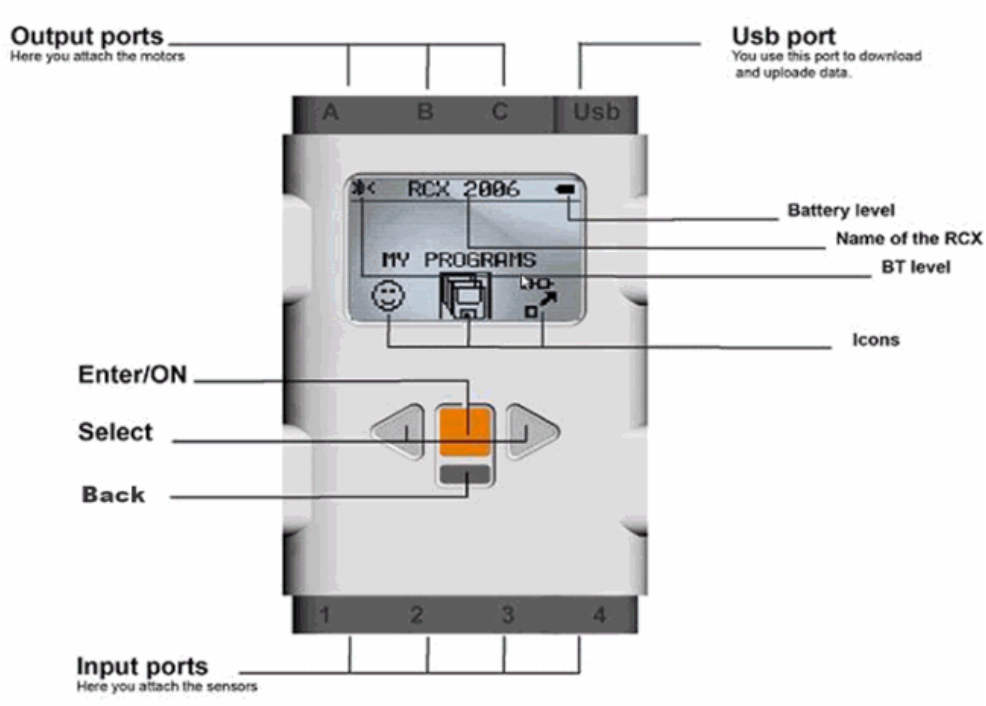


Figura 4.2.: Esquema detalle bloque NXT

Las especificaciones técnicas de la unidad de control son las siguientes:

- Microcontrolador 32-bit ARM7 (256 Kbytes FLASH, 64 Kbytes RAM).
- Microcontrolador 8-bit AVR (4 Kbytes FLASH, 512 Byte RAM).
- Bluetooth wireless communication (compatible con Clase II v 2.0).
- USB version 2.0 full speed port (12 Mbit/s).
- 4 puertos de entrada, 6-wire cable digital platform (uno de ellos compatible con IEC 61158 Type 4/EN 50 170).
- 3 puertos de salida, 6-wire cable digital platform.
- Display LCD 100 x 64 píxel.
- Altavoz (8 kHz).
- Alimentacion: 6 pilas AA o una batería de litio recargable.

En la siguiente figura se puede observar la arquitectura interna de la unidad de control:

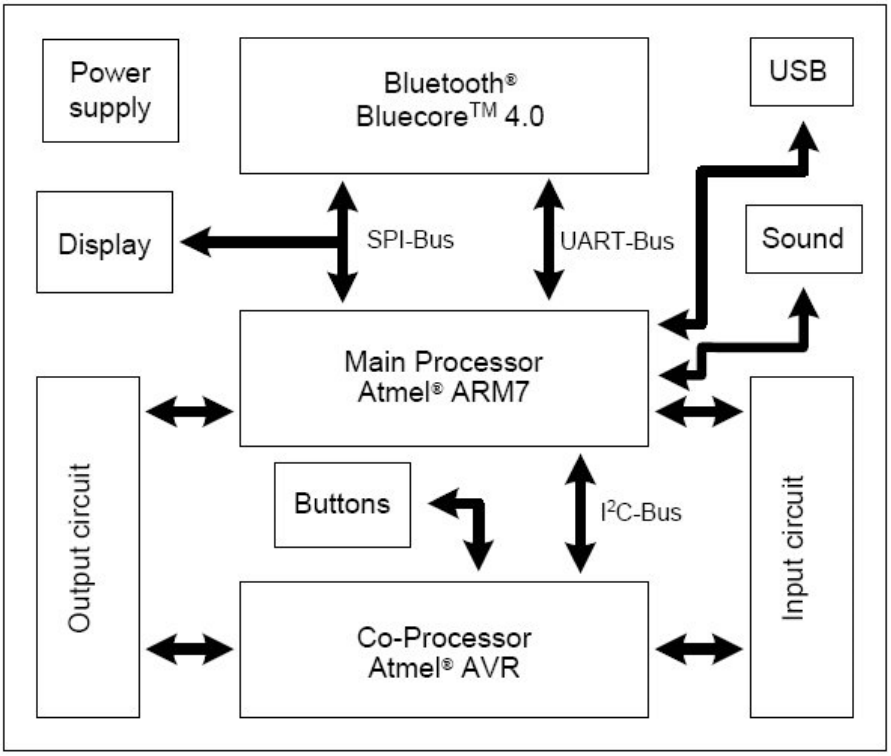


Figura 4.3.: Arquitectura interna bloque NXT

4.2.2. MOTORES

El bloque servomotor está formado por un motor eléctrico de corriente continua, un tren de engranajes o caja reductora y un sensor óptico de rotación.

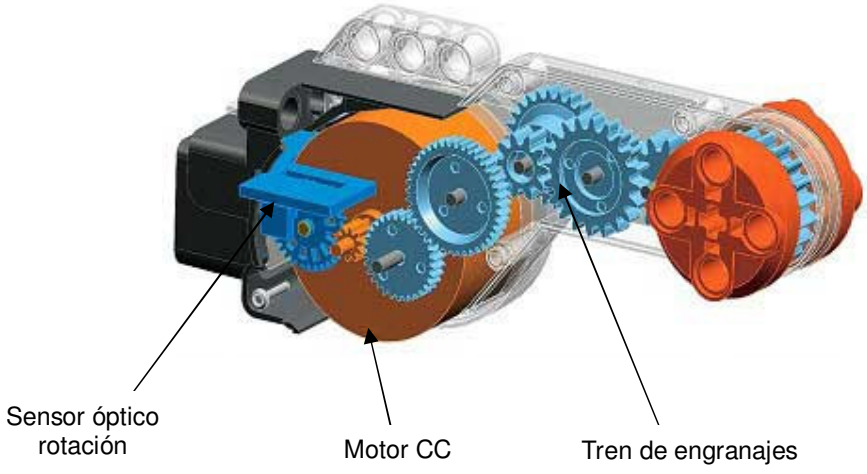


Figura 4.4.: Representación interna bloque servomotor

Los servomotores con los que viene equipado el robot Mindstorms, a diferencia de un motor eléctrico convencional, incorporan un sensor de rotación que permite el control de los movimientos del robot con alta precisión ya que, en todo momento, se tiene información relativa a la posición del motor. El sensor de rotación es capaz de medir la posición del motor con una exactitud de ± 1 grado. Una rotación completa equivale a 360 grados.

Figura 4.5.: Detalle del sensor de posición

El giro del motor es demasiado rápido para ser conectado directamente a las ruedas u otras cargas, por ello es necesario el uso de un tren de engranajes que adecua la velocidad del motor y permite que la transmisión del par sea mayor. Esto también provoca que el tamaño del motor sea mayor.

Figura 4.6.: Bloque Servomotor

4.2.3. SENSORES

Como se dijo en el capítulo 2, los sensores aportan al sistema información relativa al entorno que le rodea y son fundamentales para la autonomía del robot. El Lego Mindstorms NXT tiene varios tipos de sensores diseñados específicamente para él y que también resultan

compatibles con su predecesor RCX. Los diferentes sensores con los que viene equipado el NXT son: ultrasonido, sonido, óptico, contacto y posición (integrado en el motor) [38].

4.2.3.1. SENSOR DE ULTRASONIDO

El sensor ultrasonido le dota al robot del sentido de la vista. Este sensor puede detectar obstáculos y medir la distancia entre el sensor y el objeto en centímetros o pulgadas, desde 0 a 255 cm con una precisión de ±3 cm. Esta funcionalidad le hace ser un sensor más complejo que el resto, ya que incorpora su propio microprocesador para el cálculo de la distancia.

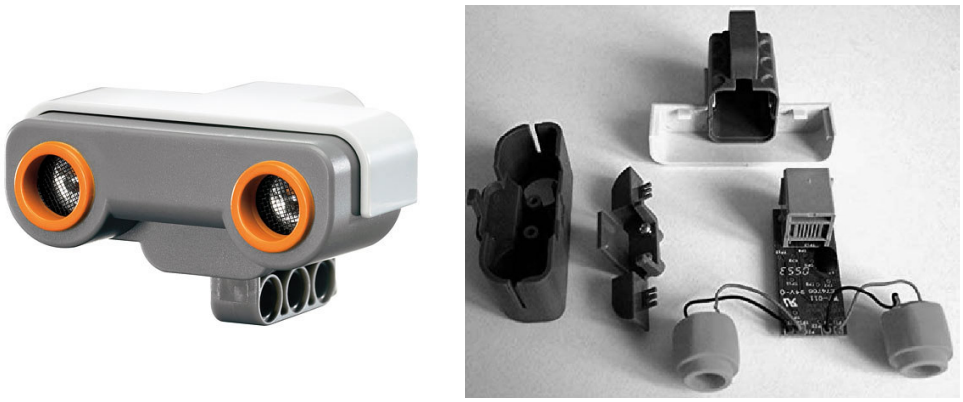


Figura 4.7.: Sensor Ultrasonido

El principio de funcionamiento de este sensor del NXT es muy similar al que se utiliza en otros muchos sistemas de medida de distancia por ultrasonidos: mide el tiempo que tarda una señal ultrasónica (f = 40KHz) en ir y volver. Si el tiempo es pequeño, el objeto está cerca; si el tiempo es grande, el objeto está lejos y, si el tiempo es mayor que un determinado valor, el objeto está fuera de alcance.

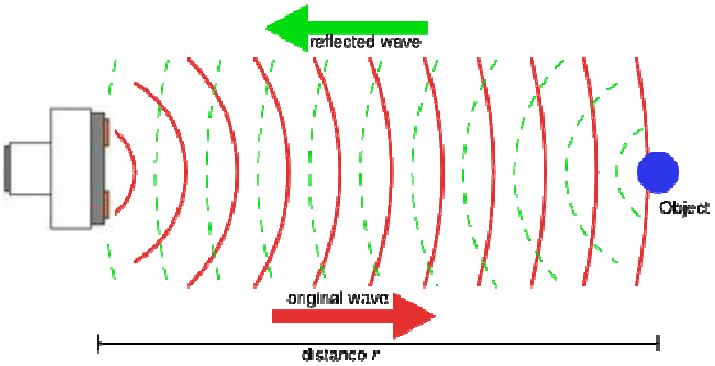


Figura 4.8.: Funcionamiento del sensor de ultrasonido

Con objetos de mayor tamaño y superficie dura obtendremos mejor rendimiento de lectura, por el contrario, los objetos hechos de tela que son curvos, delgados o pequeños son más difíciles de detectar.

La actividad de forma simultánea de varios dispositivos ultrasonido en un mismo recinto puede interferir en el correcto funcionamiento del sensor.

4.2.3.2. SENSOR DE SONIDO

El sensor de sonido dota al robot del sentido del oído. Este sensor es capaz de detectar el nivel de ruido en decibelios [dB] y decibelios ajustados [dBA].

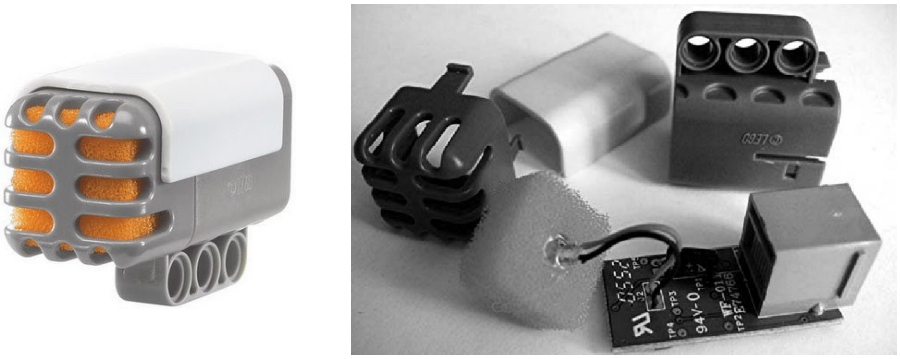


Figura 4.9.: Sensor de Sonido

Función dBA: La sensibilidad del sensor está adaptada a la sensibilidad del oído humano. Se mide el nivel de ruido (presión sonora) en la escala de ponderación A, mediante la cual, el sonido que recibe el sensor es filtrado con el objetivo de quitar parte de las muy bajas y muy altas frecuencias.

Función dB: En esta modalidad, se prescinde del filtrado y todas las frecuencias se reciben con la misma sensibilidad. Algunas de estas frecuencias son muy bajas o muy altas, por lo tanto, inapreciables para el oído humano.

Este dispositivo es capaz de realizar medidas de sonido de hasta 90 dB. Las lecturas se muestran en tanto por ciento y de acuerdo a la siguiente escala de intensidad:

PORCENTAJE	NIVEL INTENSIDAD
4% - 5%	Muy bajo
5% - 10%	Bajo
10% - 30%	Medio
30% - 100%	Alto / muy alto

Tabla 4.1.: Relación nivel de intensidad - porcentaje

4.2.3.3. SENSOR ÓPTICO

Al igual que el sensor de ultrasonido, el sensor óptico le proporciona al Lego Mindstorms el sentido de la vista. Éste está compuesto por un diodo emisor de luz (LED) y de un fototransistor.

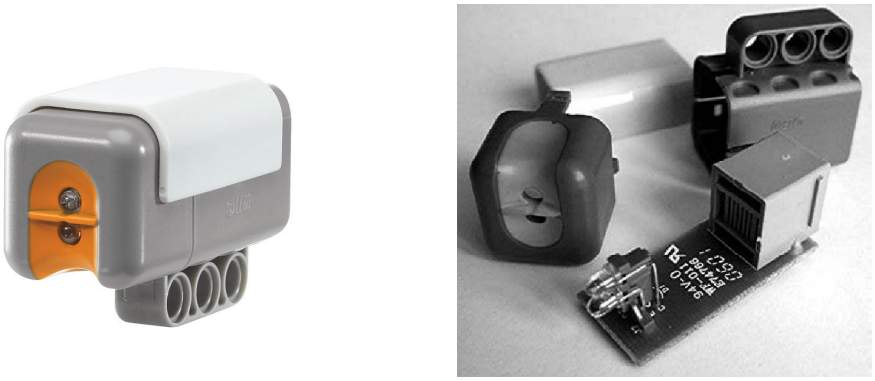


Figura 4.10. : Sensor óptico

El sensor presenta dos modos de funcionamiento:

1. Modo medidor de nivel: Capacidad de distinguir entre claridad y oscuridad, realizando lecturas del nivel de luz de su entorno en escala de 0 – 100 (oscuridad total – muy luminoso).

2. Modo reflexión: Capacidad de distinguir la intensidad de color en una superficie. El propio sensor emite una luz y después mide el porcentaje de luz reflejado.

Cómo se puede observar en la figura 4.11., el fototransistor que incorpora el sensor óptico resulta más sensible a los colores de luz infrarroja que a los colores perceptibles por el ojo humano.

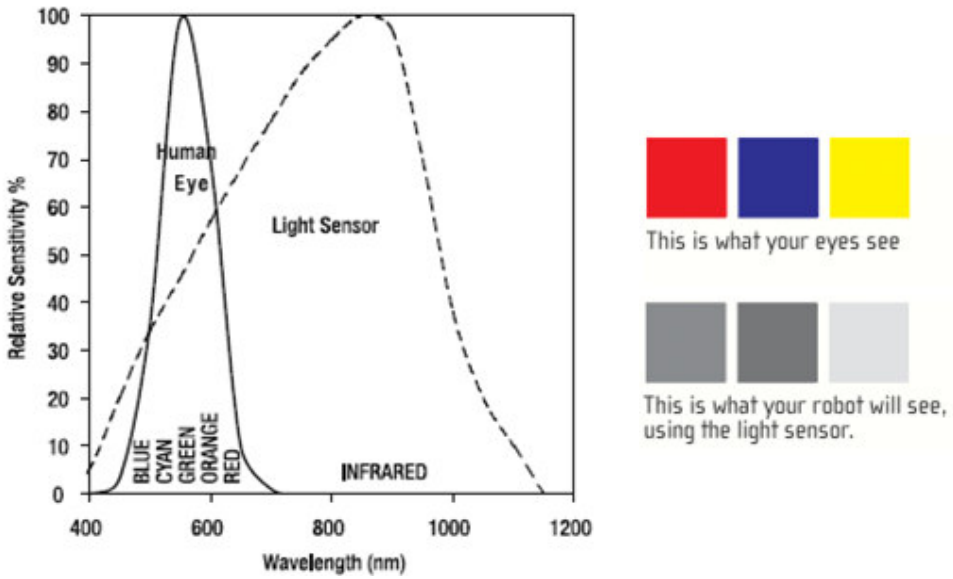


Figura 4.11. : Respuesta espectral y ejemplo de percepción de la gama de colores

4.2.3.4. SENSOR DE CONTACTO

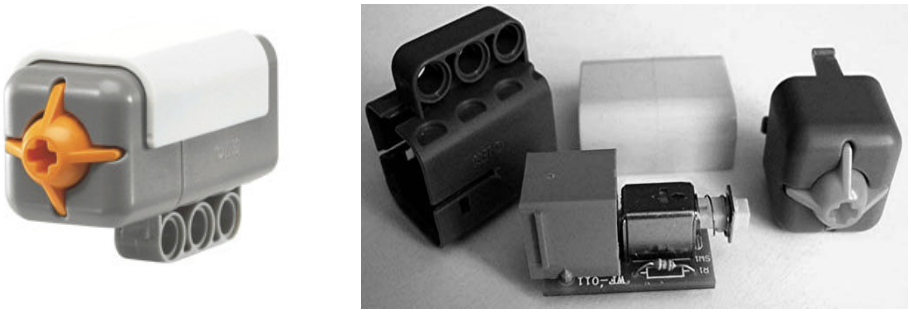


Figura 4.12. : Sensor de Contacto

El sensor de contacto le proporciona al robot el sentido del tacto. Este detecta cuando se ejerce una presión sobre él y cuando deja de producirse.

Los eventos posibles son:

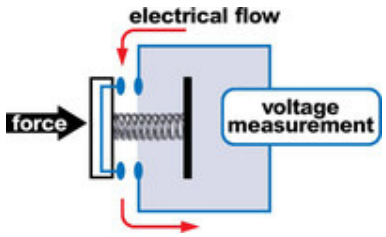


Figura 4.13.: Modos de funcionamiento sensor de contacto y representación eléctrica

4.2.4. CABLE DE CONEXIÓN

Conecta físicamente los diferentes dispositivos, motores y sensores, con la unidad de control. Representa el medio por donde se transporta el flujo de información.

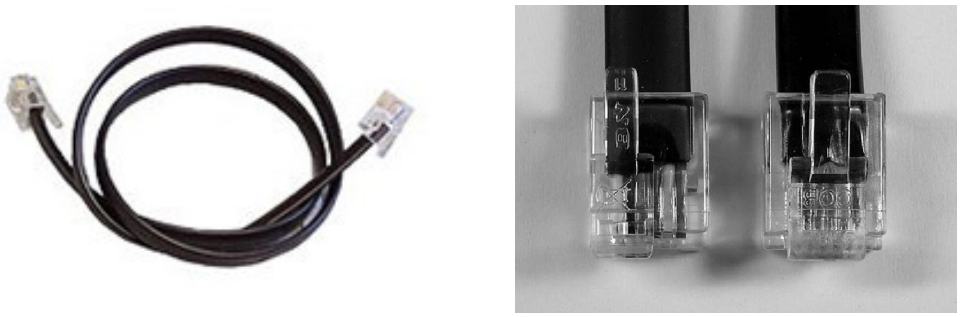


Figura 4.14.: Cable de conexión y detalle del conector

A primera vista, puede parecer un cable telefónico de 6 hilos con el correspondiente conector comercial RJ12; sin embargo, la pestaña del conector está situada en un extremo en vez de en el centro. Probablemente, esta diferencia responde a una estrategia comercial por parte del fabricante Lego, aunque se pueden encontrar gran número de tutoriales que explican los pasos a seguir para su fabricación de forma manual [38].

PIN NUMBER	COLOR	OUTPUT	INPUT
1	White	M1	AN
2	Black	M2	GND
3	Red	GND	GND
4	Green	4.3V POWER	4.3V Power
5	Yellow	TACHO0	DIGI0
6	Blue	TACHO1	DIGI1

Tabla 4.2.: Relación de colores y nombres en los pines de entrada y salida

Los cables están codificados por colores: blanco, negro, rojo, verde, amarillo y azul. Su función depende de si se utilizan en una entrada de sensor o una salida de motor.

4.3. DISEÑO DEL ROBOT MÓVIL

Los aspectos constructivos en el diseño del robot móvil vienen determinados, principalmente, por la función a desempeñar. La conclusión de los objetivos fijados está condicionada, en un alto porcentaje, por un buen diseño que permita un comportamiento estable y eficaz con los que se obtengan resultados satisfactorios en diferentes escenarios.

Existen otros factores que pueden resultar un condicionante en el diseño del robot. Por su importancia, cabe destacar la necesidad del conocimiento del medio por el que se va a desplazar, ya que ésto repercute directamente en los elementos que forman el sistema motriz del robot.

Para nuestro proyecto, se ha construido un robot móvil siguiendo un concepto de vehículo terrestre. En la siguiente figura, se pueden observar las diferentes partes relevantes del diseño:

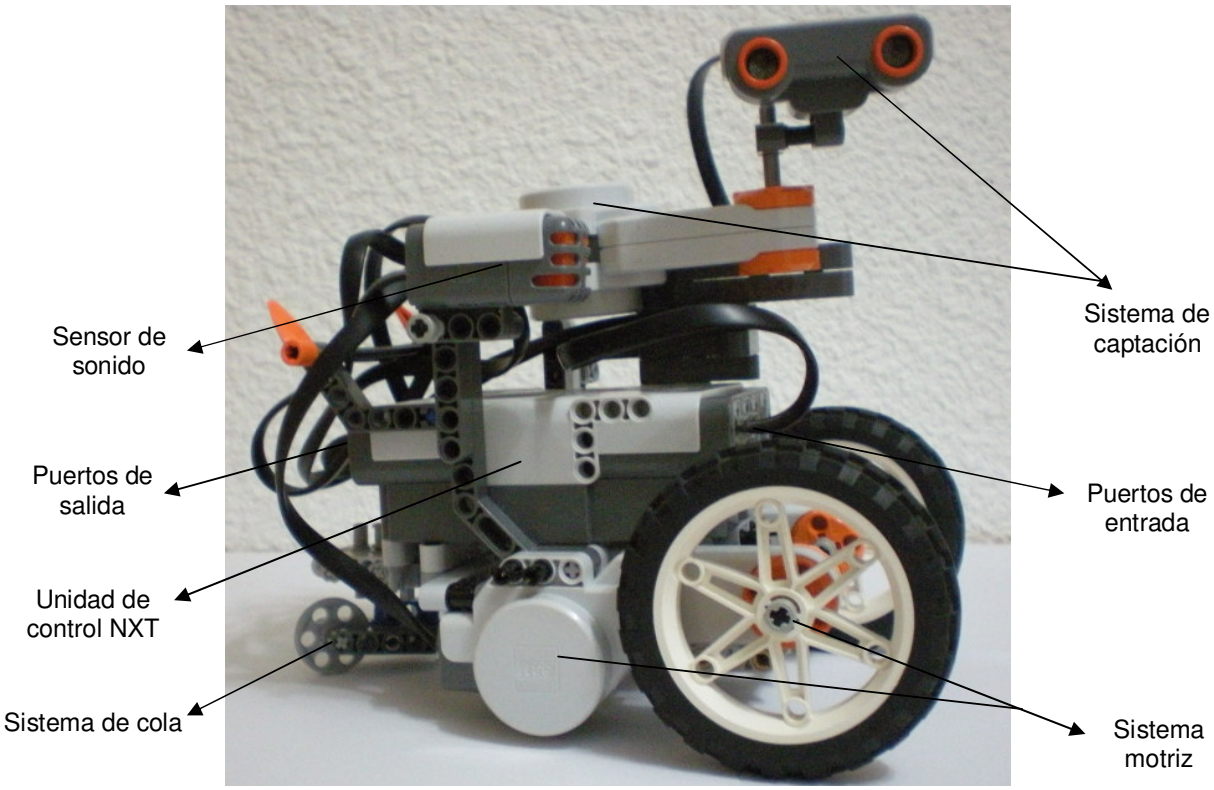


Figura 4.15.: Diseño del robot Lego Mindstorms

- Unidad de control NXT: Procesador del robot.

- Sistema de captación: Conjunto formado por el sensor ultrasonido, Sensor 1, y un servomotor, Motor A, al que se mantiene unido mediante un eje.

- Sistema motriz: Conjunto formado por dos ruedas y dos servomotores, Motor B y Motor C, al que se encuentran unidos a través de un eje.

- Sistema de cola: Su elemento fundamental es una rueda que no tiene motricidad, sin embargo, es imprescindible para comandar los movimientos de giro del robot. La precisión y efectividad de los giros están supeditados a un correcto diseño de este elemento [39]. De forma suplementaria, dota al robot de mayor robustez, ya que es la encargada de soportar todo el peso de la parte trasera del robot.

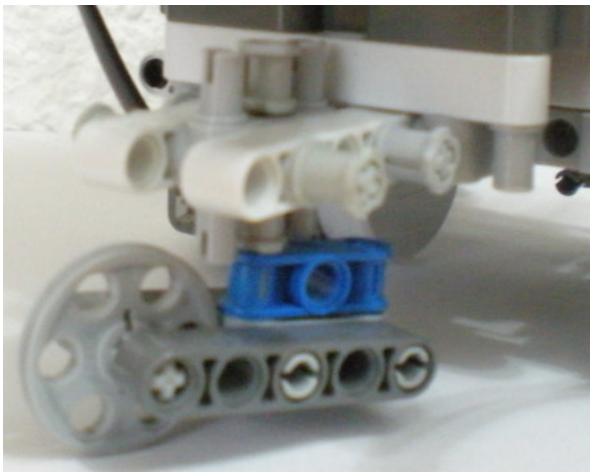


Figura 4.16.: Detalle diseños sistema de cola

- Sensor de sonido: Su cometido es captar la intensidad de sonido, representa el Sensor 2 del conjunto.

- Puertos de entrada: Representa la entrada del flujo de información proveniente de los sensores a la unidad de control.

    - ✓ Puerto 1: Sensor ultrasonido
    - ✓ Puerto 2: Sensor de sonido

- Puertos de salida: Representa la salida del flujo de información desde la unidad de control a los motores.

    - ✓ Puerto A: Servomotor del sistema de captación
    - ✓ Puerto B: Servomotor del sistema motriz
    - ✓ Puerto C: Servomotor del sistema motriz

4.4. PROGRAMACIÓN DEL ROBOT MÓVIL

4.4.1. LENGUAJES DE PROGRAMACIÓN COMERCIALES

4.4.1.1. ROBOLAB 2.9

Robolab es la herramienta gráfica que tradicionalmente se ha utilizado con los sistemas de robótica basados en Lego Mindstorms RCX. Esta herramienta se basa en el lenguaje de programación gráfico LabVIEW, desarrollado por la empresa National Instruments, ampliamente difundido en el mundo científico y tecnológico.

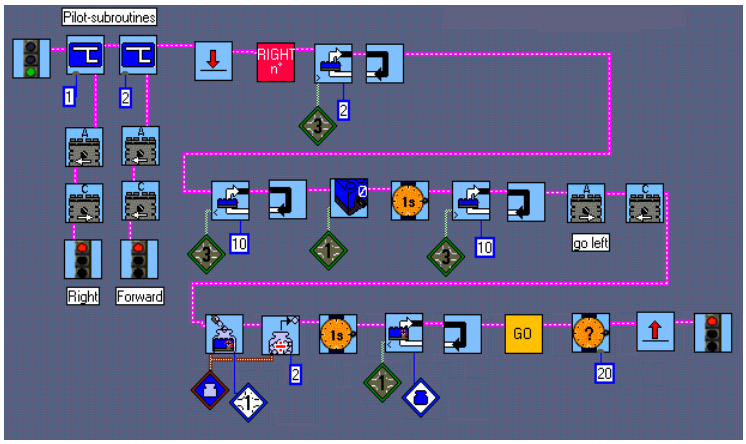


Figura 4.17.: Diagrama de flujo en Robolab 2.9

Robolab 2.9 fue concebida como una versión de transición para que los usuarios del Lego Mindstorms RCX pudiesen migrar con facilidad hacia el nuevo bloque NXT, ya que permite programar ambos bloques.

4.4.1.2. NXT-G

Sustituye a Robolab como software original de LEGO dirigido tanto al sector educativo como a la versión comercial. Al igual que Robolab, está basado en LabVIEW y ha sido desarrollado a partir de la colaboración de National Instruments y LEGO. Es un software gráfico, en el que se programa por medio de secuencias de bloques y que nos da como resultado una programación sencilla e intuitiva.

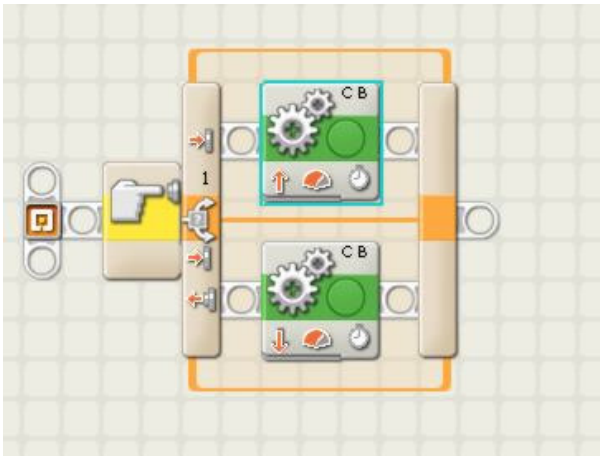


Figura 4.18.: Diagrama de flujo en NXT-G

4.4.1.3. ROBOT C

Desarrollado por Robotics Academy (Carnegie Mellon University) es un software dirigido al ámbito educativo, cuya principal característica es que permite programar en lenguaje C, además de incluir un entorno de programación (IDE) tanto para escribir como para depurar los programas. De hecho, es el único lenguaje a este nivel que ofrece un depurador en tiempo real.

Este software puede utilizarse tanto con el bloque NXT como con el bloque RCX. Al contrario del software original de Lego, RobotC da soporte al bus I2C del NXT, dispone de data-logging, punto flotante, comunicaciones Bluetooth, eventos y multitarea.

Como punto a favor de este lenguaje frente a otras opciones, se puede destacar, aparte de las citadas anteriormente, el hecho de que es, con diferencia, el de ejecución más rápida. Si necesitamos trabajar con aplicaciones que requieran gran velocidad de ejecución, robotC debe ser muy tenido en cuenta. La principal desventaja es que, actualmente, sólo es compatible con Windows.

4.4.1.4. MATLAB Y SIMULINK

MATLAB es un software que posee un lenguaje de programación de alto nivel para cálculo numérico, de adquisición de datos y análisis. Se puede utilizar para controlar robots NXT de LEGO través de un puerto serie Bluetooth (comunicación del puerto serie es parte de la funcionalidad básica de MATLAB) o

mediante una conexión USB, utilizando para ello la toolbox RWTH - Mindstorms NXT (libre y de código abierto).

Simulink es un entorno de MATLAB basado para el modelado y simulación de sistemas dinámicos. Utilizando Simulink, un usuario puede diseñar algoritmos de control, generar automáticamente código C para los algoritmos y descargar el código compilado en el NXT de LEGO.

4.4.2. LENGUAJES DE PROGRAMACIÓN DE LIBRE USO

4.4.2.1. LEJOS NXJ

Representa un reemplazo del firmware original del NXT que incluye una máquina virtual de Java, que nos permite la programación del bloque NXT de Lego Mindstorms en lenguaje Java, funcionando tanto en Windows como en Linux.

Como esta herramienta es un reemplazo de firmware, todos los archivos del firmware original del Lego Mindstorms serán sustituidos, pudiéndose recuperar, en todo momento, mediante la instalación de software suministrado por Lego.

LeJOS es un proyecto open source que fue creado originalmente por el TinyVM, siendo José Solórzano su principal representante, quien implementó una máquina virtual Java para el sistema LEGO Mindstorms RCX. La versión RCX de LeJOS ha demostrado unas cuotas de popularidad muy elevadas. El NXT ha dado al equipo de desarrollo LeJOS la oportunidad de ampliar considerablemente la capacidad y eficiencia de este software.

4.4.2.2. BRICX CC

Bricx Command Center es un conocido entorno de desarrollo integrado (IDE sus siglas en inglés) que soporta la programación con C, C++, Pascal, Forth, y Java para el bloque RCX y desarrollar y compilar programas en NBC y NXC para el bloque NXT sin necesidad de cambio de firmware.

NBC (Next Byte Codes): Es un lenguaje dirigido a programadores con una síntesis de lenguaje ensamblador.

NXC (Not eXactly C): Lenguaje de alto nivel similar a C.

4.4.3. LENGUAJES DE PROGRAMACIÓN COMERCIALES DE USO GRATUITO

4.4.3.1. LABVIEW

Otra posible opción para programar el NXT Lego Mindstorms es directamente utilizando LabVIEW. Este conocido software es una herramienta gráfica para pruebas, control y diseño mediante la programación. El lenguaje que usa se llama lenguaje G, donde la G simboliza que es lenguaje Gráfico.

Labview fue creado por National Instruments (1976) para funcionar sobre máquinas MAC, salió al mercado por primera vez en 1986. Ahora está disponible para las plataformas Windows, UNIX, MAC y GNU/Linux.

Como se ha dicho, es una herramienta gráfica de programación, esto significa que los programas no se escriben, sino que se dibujan, facilitando su comprensión. Al tener ya prediseñados una gran cantidad de bloques, se le facilita al usuario la creación del proyecto, en vez de invertir una gran cantidad de tiempo en programar un dispositivo/bloque.

Si se dispone de la aplicación "LabVIEW Toolkit para LEGO MindStorms NXT" es posible desarrollar nuevos bloques de programación para NXT-G.

Si no se dispone de LabVIEW, National Instruments facilita la obtención del sotfware LabVIEW 7.1 versión estudiante de modo gratuito. La licencia limita su uso como software de desarrollo para el NXT.

4.4.3.2. MICROSOFT ROBOTICS STUDIO

Se trata de un software que incluye un entorno de programación visual. También permite hacer simulaciones en 3D. Está dirigido tanto a educación, aficionados como a desarrolladores comerciales. Soporta diferentes plataformas, entre ellas LEGO RCX y NXT, KUKA, Fischertechnik… Está publicada la versión 1.0 y Microsoft ofrece una licencia gratuita si su uso no es comercial.

4.5. TOOLBOX RWTH-MINDSTORMS NXT

La toolbox RWTH-Mindstorms NXT para MATLAB, ha sido desarrollada para el control del robot Lego Mindstorms NXT usando MATLAB a través de una conexión inalámbrica Bluetooth o vía USB. Este software es un producto gratuito de código abierto y está sujeto a la GNU General Public License (GPL) [42]. El desarrollo de esta aplicación, fue motivada por un proyecto académico de la Universidad RWTH de Aachen; en consecuencia, su diseño está orientado principalmente para fines educativos.

Las funciones de la toolbox se basan en el protocolo de comunicación LEGO Mindstorms NXT Bluetooth para controlar el ladrillo inteligente NXT a través de una conexión inalámbrica Bluetooth o vía USB.

La principal ventaja de este concepto de control remoto es que permite combinar aplicaciones de robot con operaciones matemáticas complejas y visualizaciones en MATLAB. Esta toolbox ofrece posibilidades ilimitadas para dotar a los robots de inteligencia artificial y otras prestaciones con las múltiples funciones de MATLAB y cálculo para el procesamiento de señales digitales.

El recopilatorio de comandos de la toolbox, se puede dividir fundamentalmente en tres grupos. Por una parte, están los comandos para establecer la comunicación Bluetooth entre el Lego NXT y MATLAB; en segundo lugar, nos encontramos los comandos dirigidos al control de los motores del robot y, por último, tenemos los comandos cuyo fin es el control de los diferentes sensores [41].

Indicar que la lista de comandos es susceptible a modificaciones o a nuevas definiciones, debido al frecuente desarrollo de actualizaciones de la toolbox. Para nuestro caso, la versión de toolbox utilizada es la 2.03. A continuación, se muestra una relación de los comandos más utilizados en el presente proyecto, clasificados atendiendo a los grupos definidos líneas arriba.

4.5.1. COMANDOS DE COMUNICACIÓN

- *COM_MakeBTConfigFile*: Crear el archivo inicial (.ini), dónde se configuran y definen los parámetros de la comunicación Bluetooth.

- *COM_OpenNXT*: Abrir conexión Bluetooth.

- *COM_CloseNXT*: Cerrar conexión Bluetooth.
- *COM_SetDefaultNXT*: Establecer los valores predeterminados de la comunicación Bluetooth.
- *COM_GetDefaultNXT*: Mostrar los valores predeterminados de la comunicación Bluetooth.

4.5.2. COMANDOS DE MOTOR

Estos comandos proporcionan un nivel muy alto de control, ya que, de forma sencilla, se accede a la mayoría de las características del motor. A la hora de trabajar con los motores, inicialmente se especificará con cuál de los motores del robot se desea trabajar, los comandos sucesivos afectarán al motor en cuestión.

Una vez configurados todos los parámetros del motor, se realizará el envío de información al motor correspondiente para que las diferentes instrucciones sean ejecutadas.

- *SetMotor:* Selecciona motor.
- *GetMotor:* Muestra qué motor está seleccionado.
- *SetPower.* Establece la potencia del motor.
- *SetAngleLimit:* Establece la distancia a recorrer.
- *SetTurnRatio:* Sentido de giro del motor.
- *SyncToMotor:* Sincroniza el movimiento de varios motores.
- *SendMotorSettings:* Envía la orden al motor con los parámetros preestablecidos.
- *GetMotorSettings:* Muestra información de los parámetros del motor en tiempo real.
- *StopMotor:* Parar motor.
- *ResetMotorAngle.* Poner a 0 el parámetro *"angle"* del motor.
- *WaitForMotor:* Esperar a que se terminen de ejecutar las órdenes enviadas a los motores.

4.5.3. COMANDOS DE LOS SENSORES

Los siguientes comandos nos permiten controlar todos los sensores del Lego NXT MIndstorms:

- *OpenSwitch:* Habilita sensor de contacto.
- *GetSwitch*: Recoge información sobre el estado del sensor de contacto.
- *OpenSound*: Habilita sensor de sonido.
- *GetSound:* Recoge información del sensor de sonido.
- *OpenLight*: Habilita sensor óptico.
- *GetLight:* Recoge información del sensor óptico.
- *OpenUltrasonic:* Habilita sensor de ultrasonido.
- *GetUltrasonic:* Recoge información del sensor ultrasonido.
- *CloseSensor:* Cierra sensor.

5. DESCRIPCIÓN DEL DESARROLLO

El presente capítulo trata, de forma específica y detallada, las diferentes etapas que se desarrollan durante la ejecución del programa principal y del algoritmo de localización.

El primer paso y fundamental, es poder habilitar un canal en el que se pueda desarrollar una comunicación e intercambio de información entre las personas, a través de un ordenador y el robot móvil.

En segundo lugar, y no menos importante, engloba la definición de todas las acciones que pretendemos que el robot móvil lleve a cabo. Nos centraremos en las subrutinas desarrolladas, con el fin de controlar los movimientos efectuados por el robot y que le permiten encontrar la salida al entorno laberíntico propuesto. Una parte muy importante de este bloque, a parte de las acciones del robot, es la adquisición y tratamiento de los datos provenientes de los sensores internos (sensor de rotación) y los sensores externos (sensor de ultrasonido). Esta información resulta indispensable para el correcto funcionamiento del algoritmo de localización utilizado.

5.1. PROGRAMA PRINCIPAL

A continuación se exponen los pasos que componen el desarrollo del programa principal durante su ejecución. De igual forma, en la figura 5.1, se muestra el diagrama que resume el funcionamiento del método:

Paso 1: Se establece la comunicación Bluetooth entre el ordenador y el robot móvil.

Paso 2: Se definen los parámetros iniciales para el algoritmo ED de localización. Tales como el tamaño de la población o el número de iteraciones deseadas.

Paso 3: Se carga el mapa del entorno del laberinto.

Paso 4: Se espera a la señal de salida correspondiente para que el robot móvil comience a operar.

Paso 5: Se ejecuta la función "chequeo".

Paso 6: Se representa el diagrama polar del entorno del robot.

Paso 7: Se ejecuta el algoritmo de evolución diferencial.

Paso 8: Se obtiene y representa en el mapa de entorno la posición y orientación inicial estimada del robot.

Paso 9: Entrada en bucle.

Paso 10: Se ejecuta la función "avance". El robot móvil avanza hasta encontrar un obstáculo.

Paso 11: Se paran todos los motores del robot.

Paso 12: Se ejecuta la función "chequeo".

Paso 13: Se representa el diagrama polar del entorno del robot.

Paso 14: Se ejecuta el algoritmo de evolución diferencial.

Paso 15: Se obtiene y representa en el mapa de entorno la posición y orientación estimada del robot.

Paso 16: Se ejecuta la función "giro".

Paso 17: El bucle, del paso 10 al paso 16, se ejecutará hasta que el robot móvil encuentre la salida del laberinto.

Paso 18: Salida del laberinto del robot móvil.

Paso 19: Parada de motores, sensores y desconexión Bluetooth.

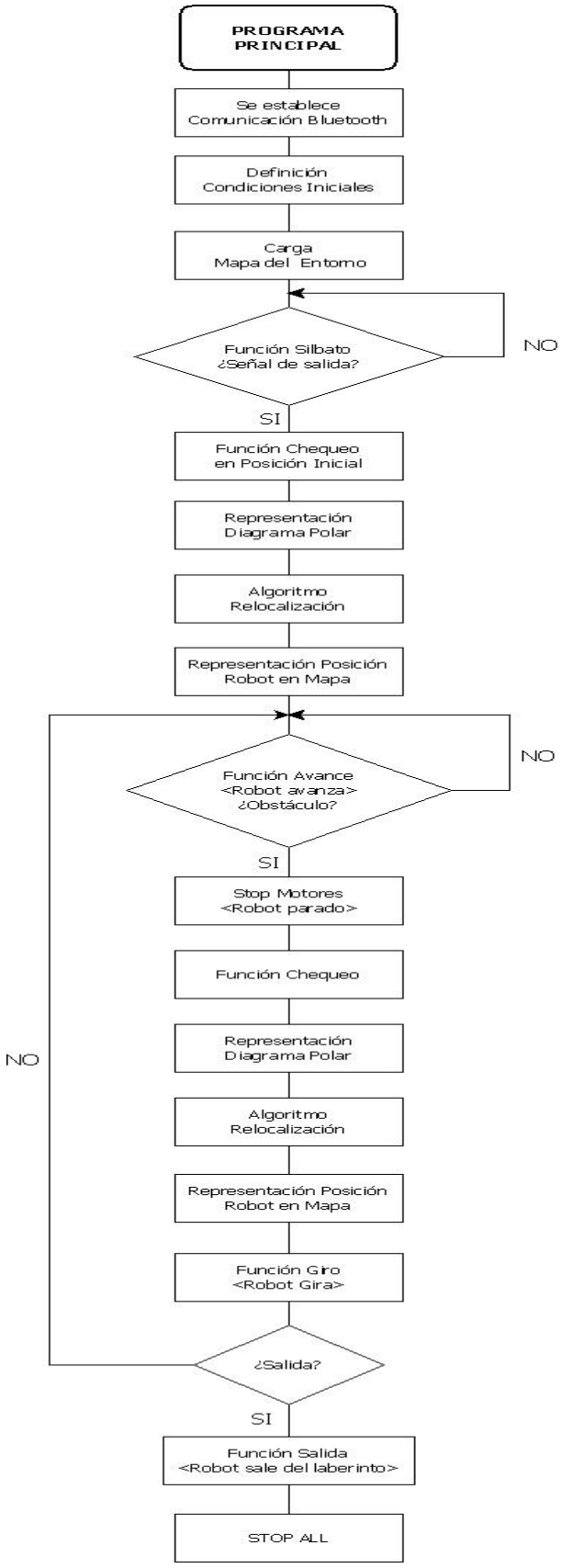


Figura 5.1: Diagrama de flujo del Programa Principal

5.2. COMUNICACIÓN PC – ROBOT MÓVIL

El robot Lego NXT Mindstorms cuenta con dos protocolos para la conexión con medios externos, tal y como se especifica en el apartado 4.2.:

- Bluetooth wireless communication (compatible con Clase II v 2.0)
- USB version 2.0 full speed port (12 Mbit/s)

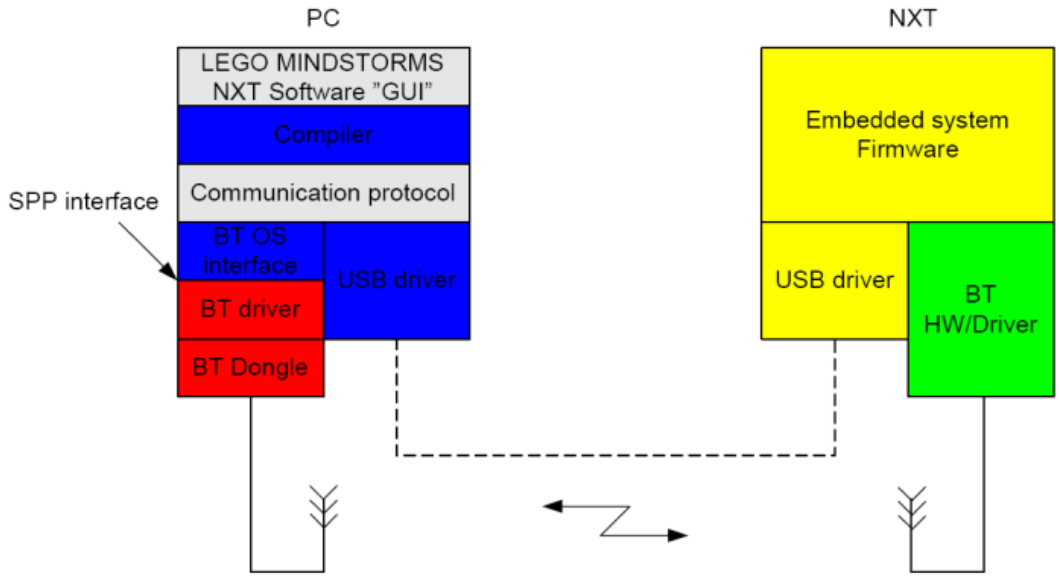


Figura 5.2.: Esquema comunicaciones Lego Mindstorms

La unidad de control NXT permite la comunicación inalámbrica a través del protocolo de comunicación Bluetooth v2.0 que incorpora la técnica "Enhanced Data Rate" (EDR), que permite mejorar las velocidades de transmisión hasta 3Mbps, mediante la inclusión de un chip CSR BlueCore 4, versión 2. Se trata, de un dispositivo Bluetooth Clase II, por lo que el consumo de energía es menor respecto a otros dispositivos Bluetooth y nos proporciona un alcance aproximado de 10 metros.

Es compatible con la tecnología Serial Port Profile (SPP), que puede ser considerado como un puerto serie inalámbrico, permitiéndonos enviar y recibir información a la unidad de control NXT durante la ejecución del programa.

Por otro lado, también es posible comunicarse a través de la interfaz USB v2.0. Las posibilidades, tanto de comunicación como funcionales a través del dispositivo USB, son las mismas que para el caso del Bluetooth, pero la comunicación vía USB implica la utilización de una conexión física entre el ordenador y el robot.

Ante la necesidad de establecer una comunicación Bluetooth entre ambos dispositivos (PC-robot) y, atendiendo a lo mencionado anteriormente, se desestima la comunicación vía USB en favor de la comunicación inalámbrica Bluetooth, dada la versatilidad y libertad de movimiento que esta última le proporciona al robot.

El dispositivo Bluetooth utilizado para el ordenador, es un nano adaptador externo USB Bluetooth 2.1 de alta velocidad, 3 Mbps EDR, de Clase I y un alcance máximo de 200m. Este dispositivo es compatible con el Protocolo de Comunicación de LEGO® Mindstorms y admite la tecnología Serial Port Profile (SPP).

Figura 5.3.: Dispositivo Bluetooth utilizado

5.2.1. CONFIGURACIÓN COMUNICACIÓN BLUETOOTH

Para comunicarse con el NXT a través de Bluetooth, tenemos que usar el SPP (puerto serie), que básicamente funciona como un puerto serie virtual. Esta es la razón por la que se puede enviar y recibir datos desde MATLAB a través de los comandos del puerto serie.

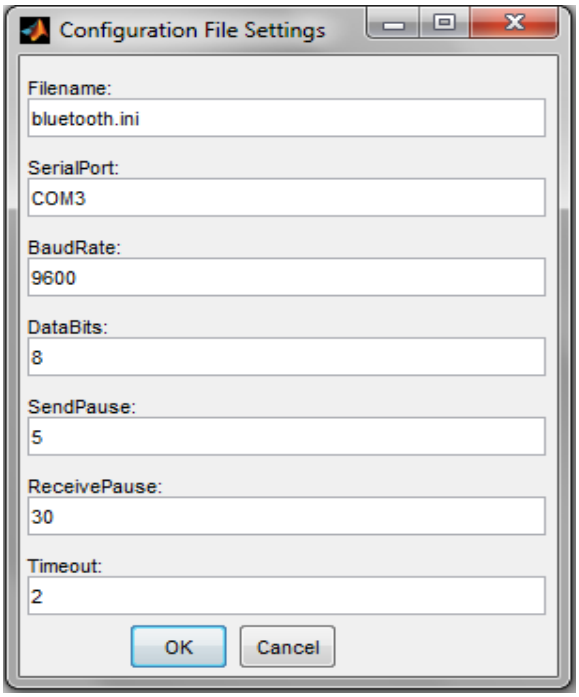


Figura 5.4.: Parámetros de configuración comunicación Bluetooth

Es imprescindible para establecer una comunicación entre el PC y el robot móvil, definir previamente unos parámetros relativos al protocolo de comunicación que incluye, entre otros, tal y como se muestra en la figura 5.4, la información referente al puerto serie utilizado, la velocidad de la comunicación o los bits de datos.

De esta forma se creará un archivo con extensión .ini al que estará referenciado el inicio de cualquier conexión Bluetooth entre el PC y el robot, permitiendo establecer una comunicación serie estable y fluida en el envío y recepción de información entre ambos [Apéndice A].

Figura 5.5.: Conexión Bluetooth establecida entre el PC y el robot

5.3. DESARROLLO DE SUBRUTINAS

En este apartado, se explica de forma detallada el funcionamiento de las diferentes subrutinas implementadas, relativas al control del robot móvil. El código fuente desarrollado se adjunta en el Apéndice A.

5.3.1. SUBRUTINA CHEQUEO

Denominaremos distancia vectorial al vector formado por las distancias que existen entre el robot y los obstáculos de su entorno en las doce direcciones consideradas principales.

El sensor responsable de la toma de medidas y que nos proporciona la distancia vectorial, es el sensor de ultrasonido del Lego Mindstorms. Como ya explicamos en el apartado 4.2.3.1, este sensor es capaz de medir la distancia entre el sensor y objetos situados como máximo a 255 cm, con una precisión de ±3 cm.

La primera dirección tomada viene determinada por la orientación del robot, a partir de la cuál se tomarán las 12 medidas siguientes con un desfase en sentido horario de 30º entre ellas hasta completar los 360º.

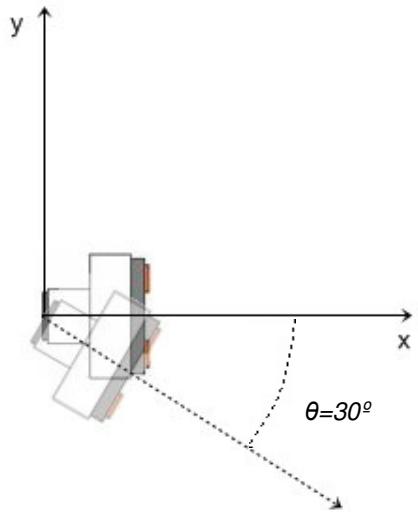


Figura 5.6.: Giro de 30º sentido horario del sensor ultrasonido

El sensor ultrasonido está unido mediante un eje a uno de los tres servomotores con los que cuenta el diseño del robot, en concreto, al motor A, que será el responsable del control de esta rotación secuencial cada 30º.

El movimiento se realizará con una relación de potencia baja, 15 sobre una escala de 100, que nos aportará una velocidad de rotación apropiada para lograr una mayor exactitud y precisión en los movimientos.

En la siguiente representación, figura 5.7, se pueden observar las diferentes posiciones que va a ocupar el sensor ultrasonido durante la ejecución de la subrutina chequeo:

Una vez finalizada la toma de medidas y, en consecuencia, determinada la distancia vectorial, el sensor ultrasonido realizará una rotación de 360º en sentido antihorario con una relación de potencia del 40% para volver a su posición de origen.

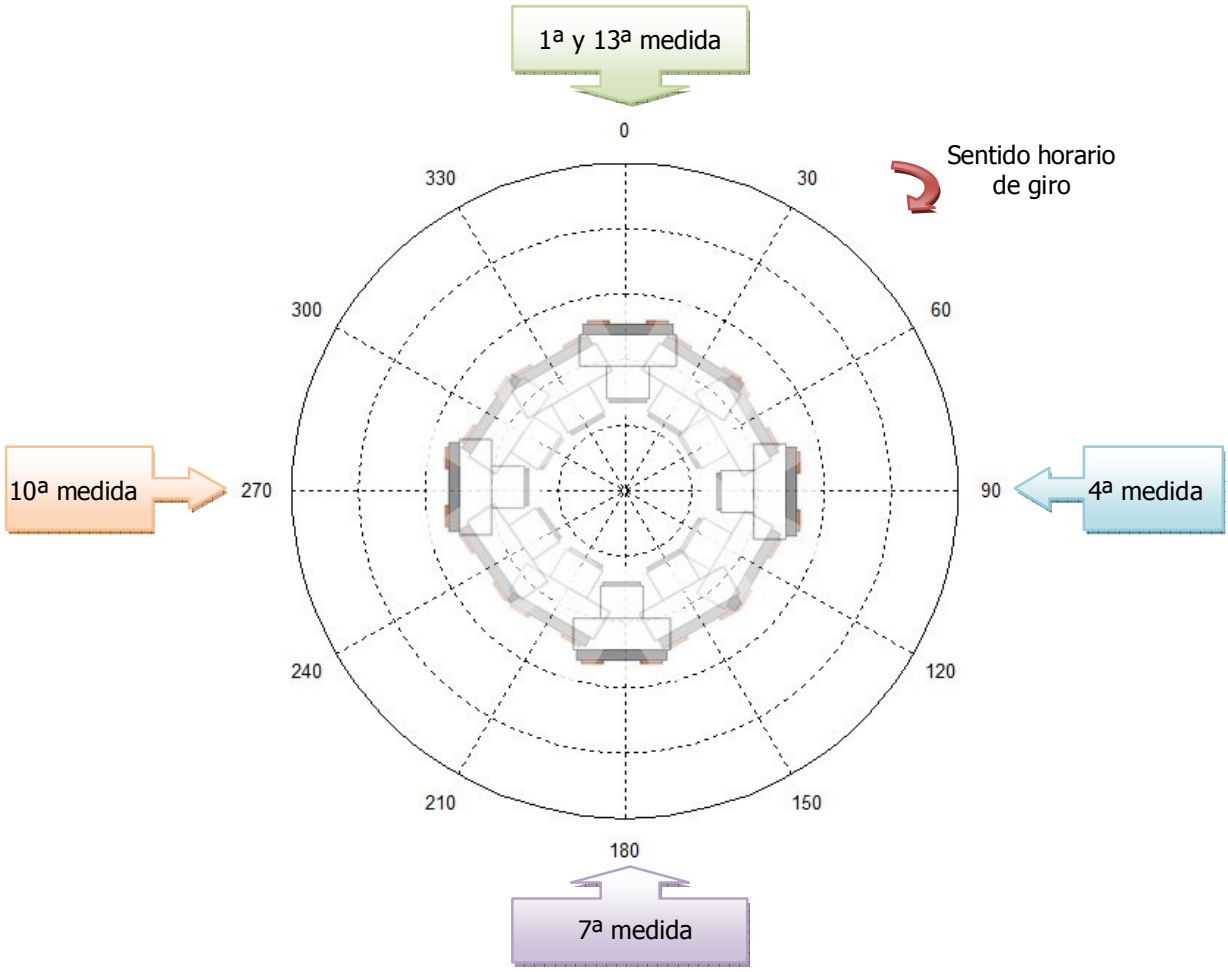


Figura 5.7.: Representación posición del sensor de ultrasonido respecto a diagrama polar

Seguidamente, obtendremos un diagrama polar, donde se representarán gráficamente las medidas obtenidas en función del ángulo ocupado respectivamente. Independientemente de la orientación absoluta del robot, la primera medida siempre corresponde al ángulo de cero grados.

Las trece medidas realizadas definen el vector distancia que resultará de vital importancia para el algoritmo de localización utilizado.

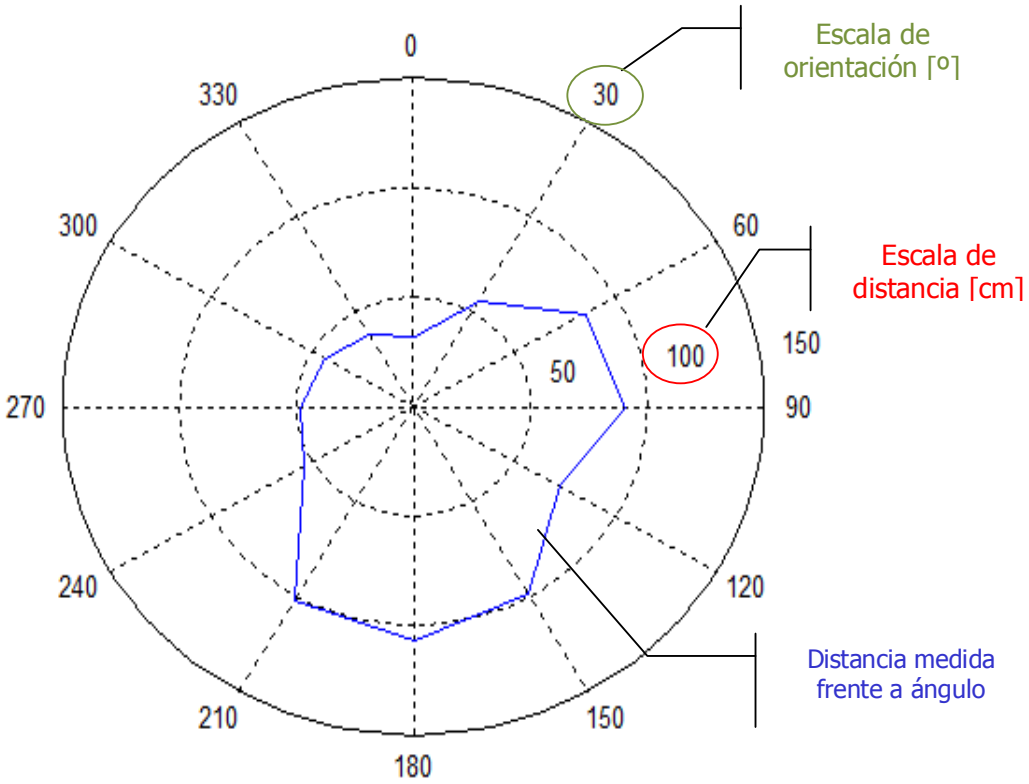


Figura 5.8.: Diagrama polar: Representación de distancia medida frente a ángulo

Como se muestra en la tabla 5.1, a cada ángulo le corresponde una medida de distancia expresada en centímetros.

Ángulo [º]	Distancia [cm]
0	32
30	56
60	85
90	91
120	72
150	99
180	107
210	102
240	54
270	48
300	44
330	38
360	32

Tabla 5.1.: Relación ángulo-distancia fig. 5.8.

En el siguiente diagrama de flujo, figura 5.9, se muestran en detalle las diferentes etapas de funcionamiento de la subrutina chequeo:

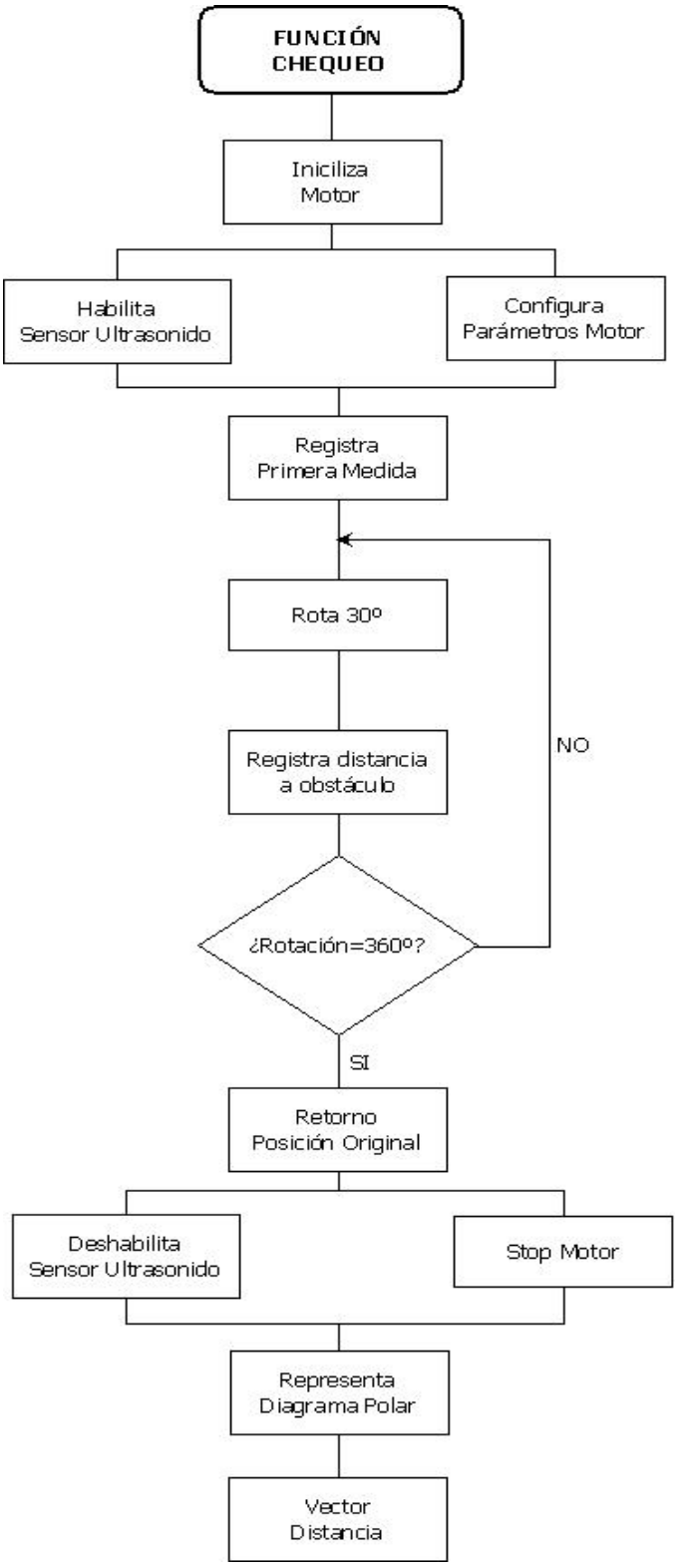


Figura 5.9: Diagrama de flujo de la subrutina Chequeo

5.3.2. SUBRUTINA AVANCE

Hasta el momento, todas las acciones realizadas por el robot, así como la recogida de datos han tenido carácter estático, ya que como acabamos de ver la ejecución de la subrutina de chequeo, no implica ningún desplazamiento por parte del robot. Como es obvio, en vista de conseguir salir del laberinto, es necesario que el robot móvil se desplace con autonomía en el entorno que se emplaza.

Para alcanzar este objetivo, se ha desarrollado la función *avance_robot.m*. En esta subrutina se desarrollan, fundamentalmente, dos aspectos:

- Se detectan los obstáculos con los que el robot se encuentra en su camino; en nuestro caso, estos obstáculos son los muros que delimitan y determinan la forma del laberinto.

- Se realiza el cálculo de la odometría obteniendo un valor expresado en centímetros que representa el desplazamiento realizado por el robot desde el punto de partida hasta la detección del obstáculo. Este dato resulta de vital importancia para la posterior relocalización del robot.

Una vez que el robot está orientado correctamente en la dirección de avance, bien desde la posición inicial o bien después de haber realizado un giro, se realiza una inicialización de los valores internos de los motores responsables del movimiento de traslación del robot, motor B y motor C. Con esta medida, evitamos que los datos referentes a movimientos anteriores queden memorizados y puedan interferir en el cálculo odométrico.

Seguidamente, se activa el sensor ultrasonido y se activan los motores para que el robot comience a avanzar. La velocidad de avance del robot será del 35 por ciento de su valor máximo. En este rango se evitan posibles deslizamientos en el arranque que puedan inducir a posibles variaciones en la dirección de desplazamiento, además de favorecer a que el número de lecturas del sensor ultrasonido sea mayor y de mayor precisión y, en consecuencia de esta velocidad moderada, la frenada del robot al detectar el obstáculo será mas suave, evitando posibles deslizamientos que puedan provocar la colisión con el muro del laberinto.

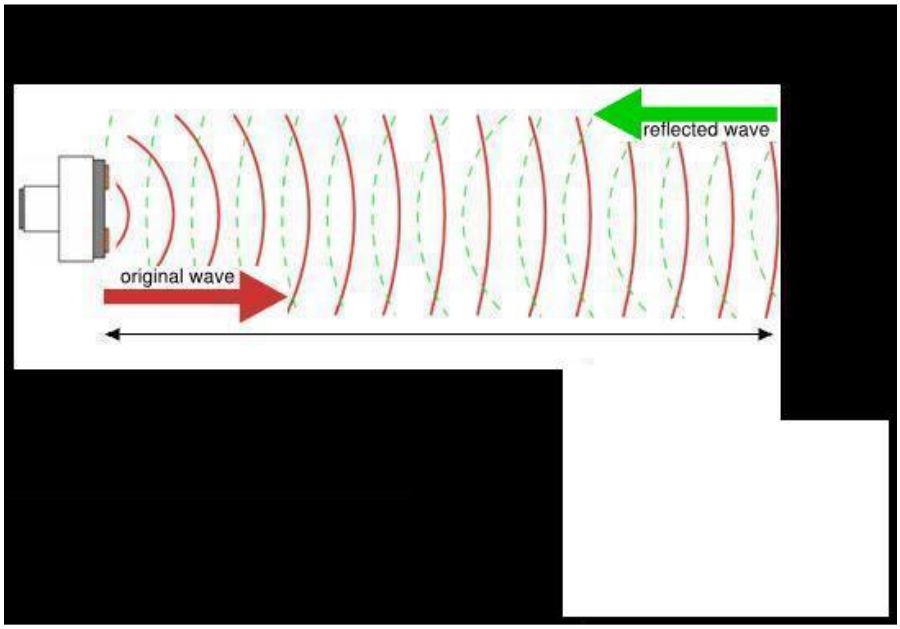


Figura 5.10.: Robot posición inicial comienza a avanzar.

De forma simultánea al avance, el robot permanecerá realizando lecturas de forma continua, por medio del sensor ultrasonido, del entorno que tiene por delante. Mientras la lectura de distancia al obstáculo más próximo, en nuestro caso de la pared del laberinto, no sea inferior a 25 cm., el robot continuará avanzando con normalidad.

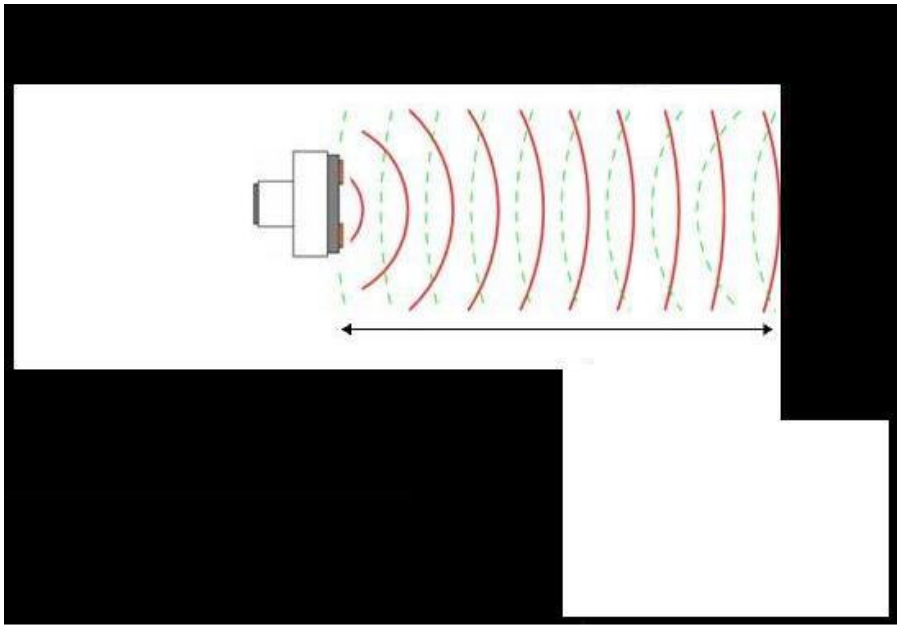


Figura 5.11.: Robot avanzando con sensor ultrasonido activo

Cuando la lectura de la distancia a la pared del laberinto sea inferior a 25 cm., el robot desactivará de forma inmediata los motores y, en consecuencia, el robot quedará frenado.

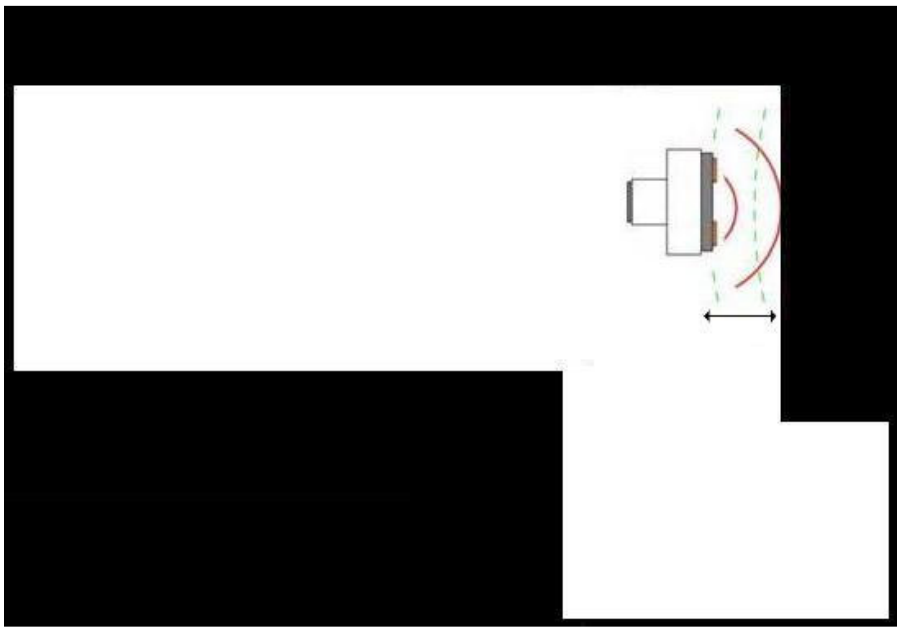


Figura 5.12.: Robot avanzando detecta obstáculo y se detiene

Llegado este punto y con el robot parado, se procede a la lectura e interpretación de la información recogida entre la que se encuentra la registrada durante el avance por los sensores odométricos alojados en sendos motores.

```
out = GetMotorSettings(MOTOR_C);

out.IsRunning          % boolean, true if the motor works
out.Power              % current power
out.AngleLimit         % current set angle limit
out.TurnRatio          % current turn ratio
out.SpeedRegulation    % boolean, speed regulated or not?
out.SyncToMotor        % the motor this one is synced to
out.TachoCount         % internal, non-resettable
out.Angle              % current motor position, resettable
out.MotorBrake         % boolean, is electronic braking enabled
```

Tabla 5.2.: Lista de parámetros internos del motor

En la tabla 5.2 se puede observar que los motores nos proporcionan información de diferente índole, para el cálculo de la odometría nos fijamos únicamente en la variable "Angle". Éste

parámetro, reseteable y expresado en grados, representa el número de vueltas realizadas por cada motor en cuestión, con una exactitud de ± 1 grado.

Para el cálculo de la odometría, resulta indispensable conocer el número de vueltas que realizan los motores desde la posición inicial hasta la posición de parada. El número de vueltas es una magnitud expresada en grados, pero nuestro objetivo final, es obtener la distancia total recorrida entre ambos puntos, por lo que será necesario realizar una conversión que nos permita trabajar con este dato expresado en centímetros.

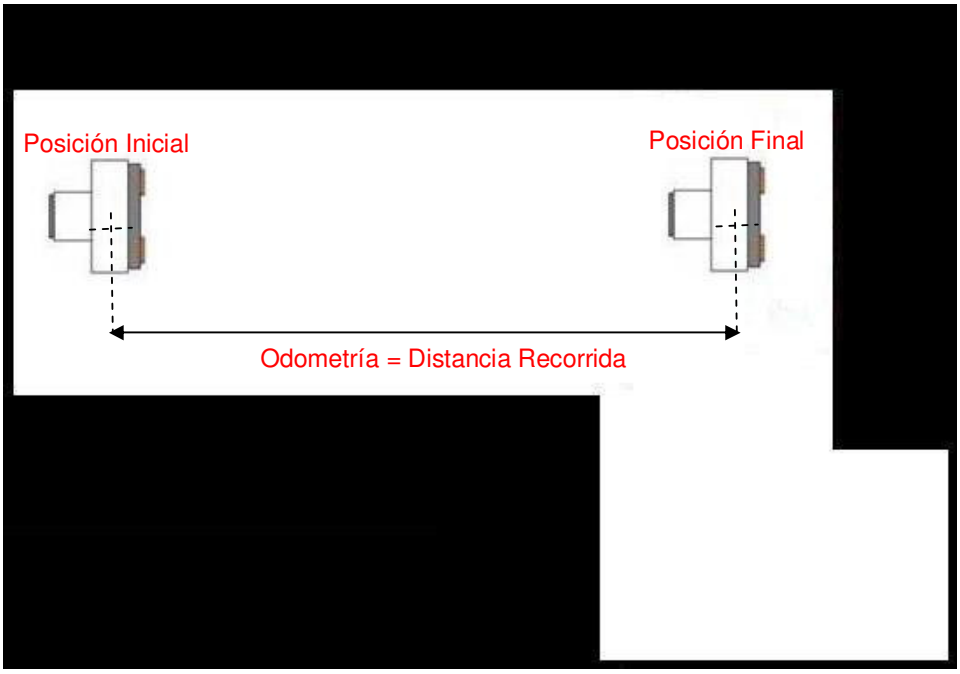


Figura 5.13.: Representación distancia recorrida por el robot móvil

Para establecer una relación entre grados y centímetros, previamente se ha tenido que realizar una calibración de los motores que nos permita determinar con la mayor exactitud posible esta equivalencia. Para ello, a partir de una distancia fija, se han realizado varias pasadas con el robot móvil y se han analizado los resultados obteniendo, en nuestro caso, la equivalencia de que 1 metro representa 1350 grados del motor.

Los dos motores que integran el sistema motriz que permite el desplazamiento del robot son independientes, motor B y C, por lo que cada uno de ellos nos proporciona sus propios valores relativos a la odometría. Esto nos asegura que, en caso de que alguna de las medidas de los sensores de posición se vea afectada por un error considerable, su

efecto se vea paliado, ya que el robot cuenta con dos medidas de referencia para realizar los cálculos de la distancia total recorrida.

A continuación, en la figura 5.14 se puede observar de forma concisa las diferentes etapas que comprende la función avance:

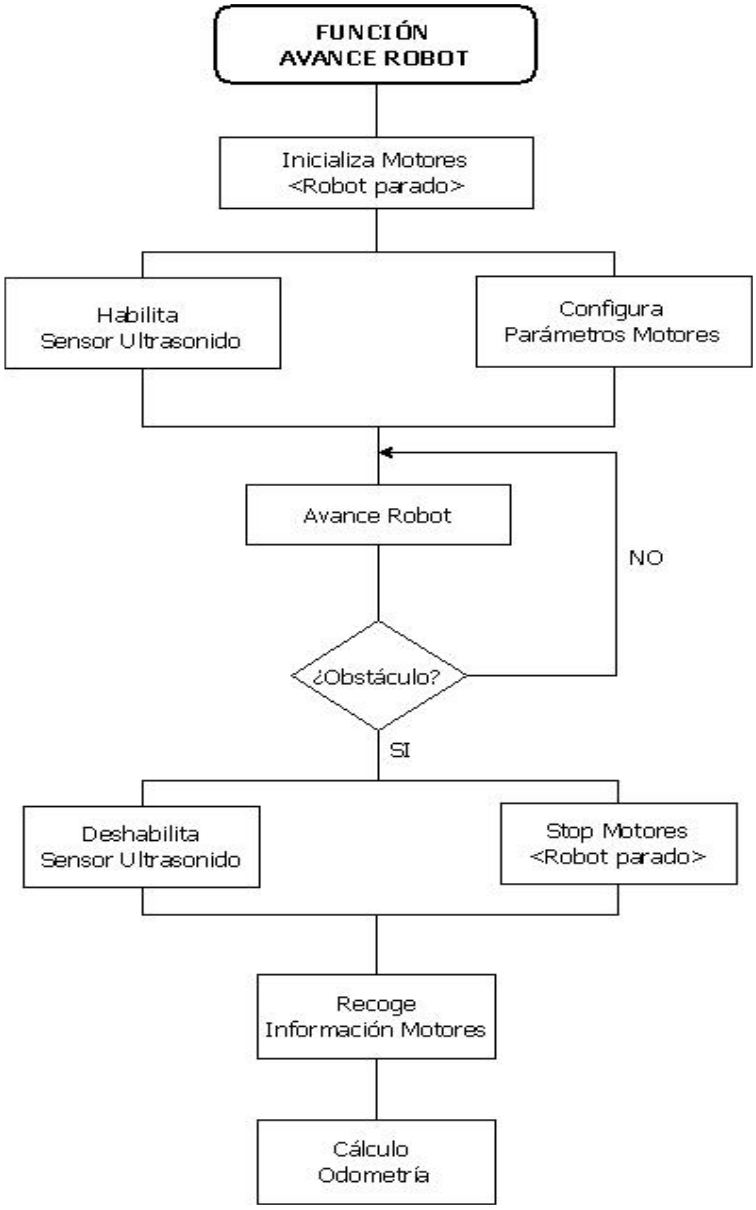


Figura 5.14: Diagrama de flujo de la subrutina Avance Robot

5.3.3. SUBRUTINA GIRO

Como es lógico, una de las principales características para que un robot móvil goce de autonomía es que éste sea capaz de realizar movimientos de giro. Con esta finalidad se desarrolla la función *giro_robot.m* en la que se definen las siguientes funciones:

- Variación de la dirección del robot, con el objetivo de sortear los obstáculos.

- Cálculo de la orientación basándose en la información odométrica, imprescindible para la posterior relocalización del robot dentro del mapa del entorno.

El objetivo más inmediato es sortear el obstáculo que impide el avance del robot, para ello es necesario hacer uso del diagrama polar, resultante del chequeo de 360º obtenido con anterioridad al ejecutar la subrutina "chequeo".

Como ya se ha mencionado, el diagrama polar es una representación gráfica de una matriz en la que se relaciona la distancia del robot al obstáculo en diferentes orientaciones o ángulos.

El criterio que nos va a definir el sentido del giro, será la distancia máxima registrada en dicho diagrama, ya que se entiende que es la opción con más probabilidades de éxito para encontrar la salida del laberinto.

Por lo tanto, tomando como referencia el diagrama polar, si la distancia máxima se encuentra entre 0º y 180º (semicírculo derecho) el robot realizará un giro en sentido antihorario y, si por el contrario, la distancia máxima se encuentra entre 0º y -180º (semicírculo izquierdo), el giro se realizará en sentido horario.

Un aspecto muy importante a la hora de la elección de la distancia máxima, es que los valores comprendidos entre 150º y 210º no se tienen en cuenta, ya que éstos representan el camino por el que ha venido el robot y podría conllevar a interpretaciones incorrectas, incluso que el robot "deshiciera" el camino recorrido.

Ángulo [º]	Distancia [cm]
0	32
30	56
60	85
90	91
120	72
150	0
180	0
210	0
240	54
270	48
300	44
330	38
360	32

Tabla 5.3.: Relación ángulo - distancia obviando la media vuelta

Tomando como ejemplo los datos de la tabla 5.3., se puede observar que, sin tener en cuenta los valores correspondientes al retroceso, la lectura de distancia mayor corresponde cuando el sensor ultrasonido se encuentra en la posición de 90º. Siguiendo la lógica de lo explicado anteriormente, este valor se encuentra en el semiplano derecho, por lo que el robot accionará su sistema motriz y girará en sentido horario, tal y como se muestra en la figura 5.15.

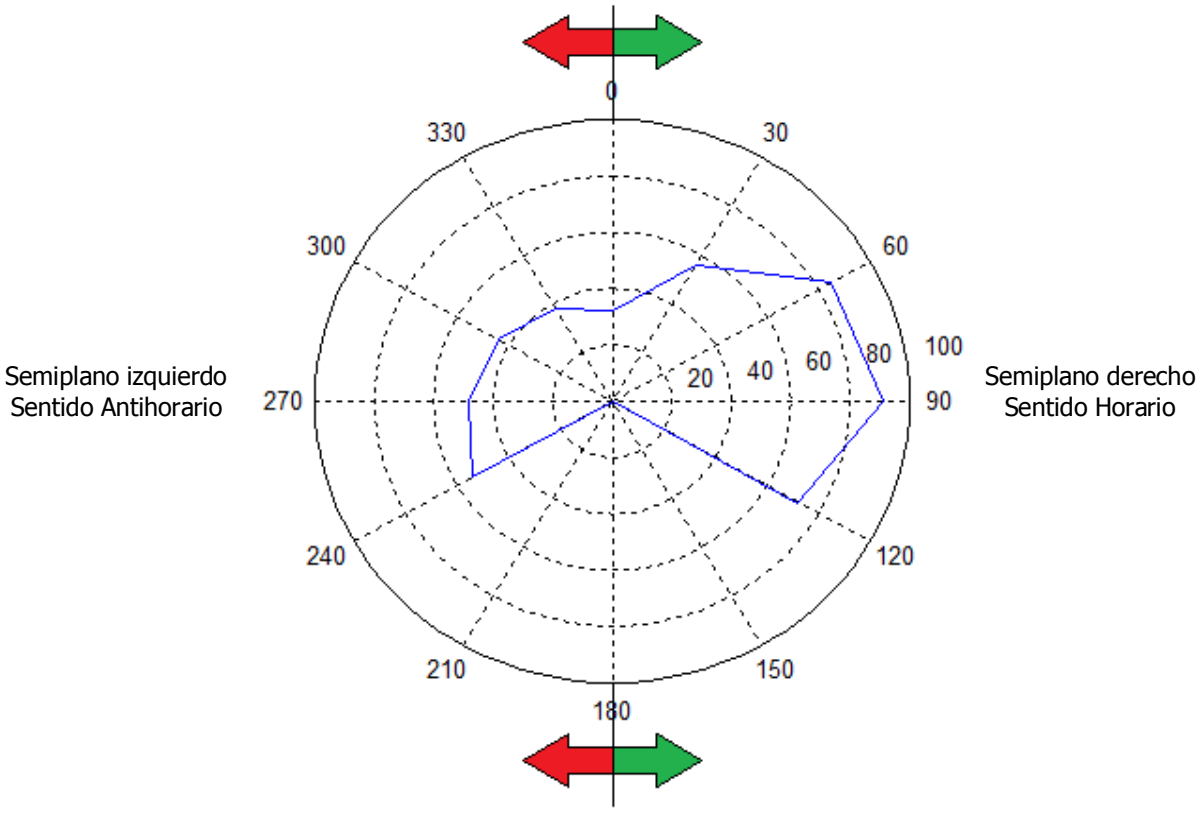


Figura 5.15.: Representación de la elección del sentido de giro

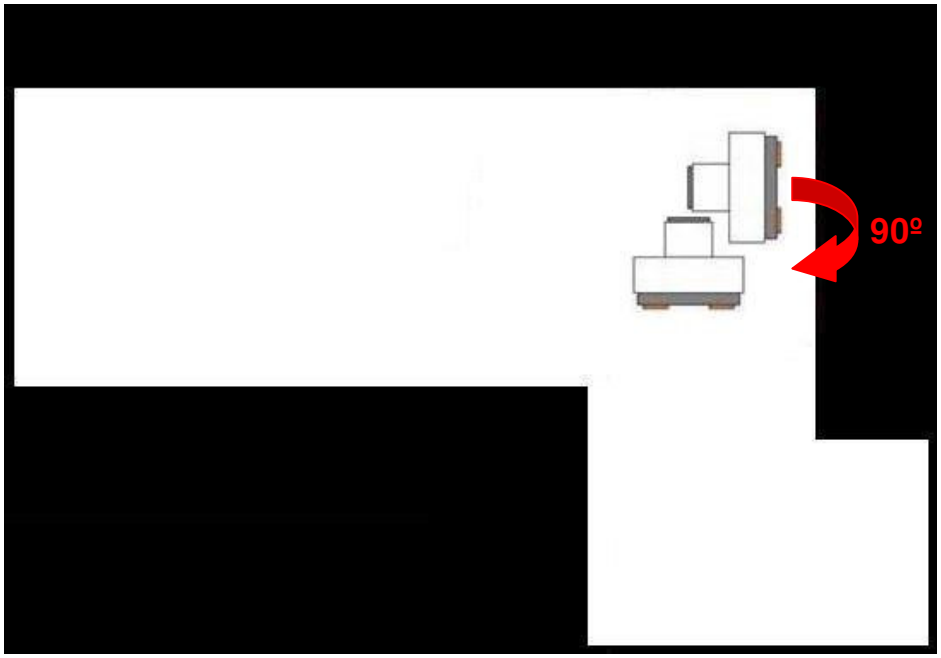


Figura 5.16.: Representación del giro realizado por el robot móvil

Al ejecutar este movimiento se produce un cambio en la orientación, evento que afecta de forma directa a la localización, ya que es uno de los parámetros a estimar. El robot está programado para que realice giros con valor de 90º y de -90º, pero estos giros de valor constante pueden ver mermada su exactitud por diferentes factores: posibles deslizamientos, textura de la superficie, nivel de baterías o agentes externos. Es por este motivo que, al igual que en el caso de la odometría en la función "avance", se accede a la información interna de los motores para saber el giro en grados realizado según la información proporcionada por los sensores de posición.

Contrastando los datos recibidos por ambos sensores, podremos saber el cambio de orientación relativo experimentado por el robot, resultando éste un parámetro fundamental para el algoritmo de localización.

Como toda medida procedente de cualquier sensor de posición, se precisa de un método de calibración para estimar la equivalencia oportuna entre los grados internos expresados como variable del motor y los grados que representan en realidad. En nuestro caso 540º de los motores del robot son 360º reales. Se puede observar que, al contrario de la odometría, aquí no es necesaria la conversión de unidades, de grados a centímetros, ya que la variable a estimar (θ) se expresa directamente en grados.

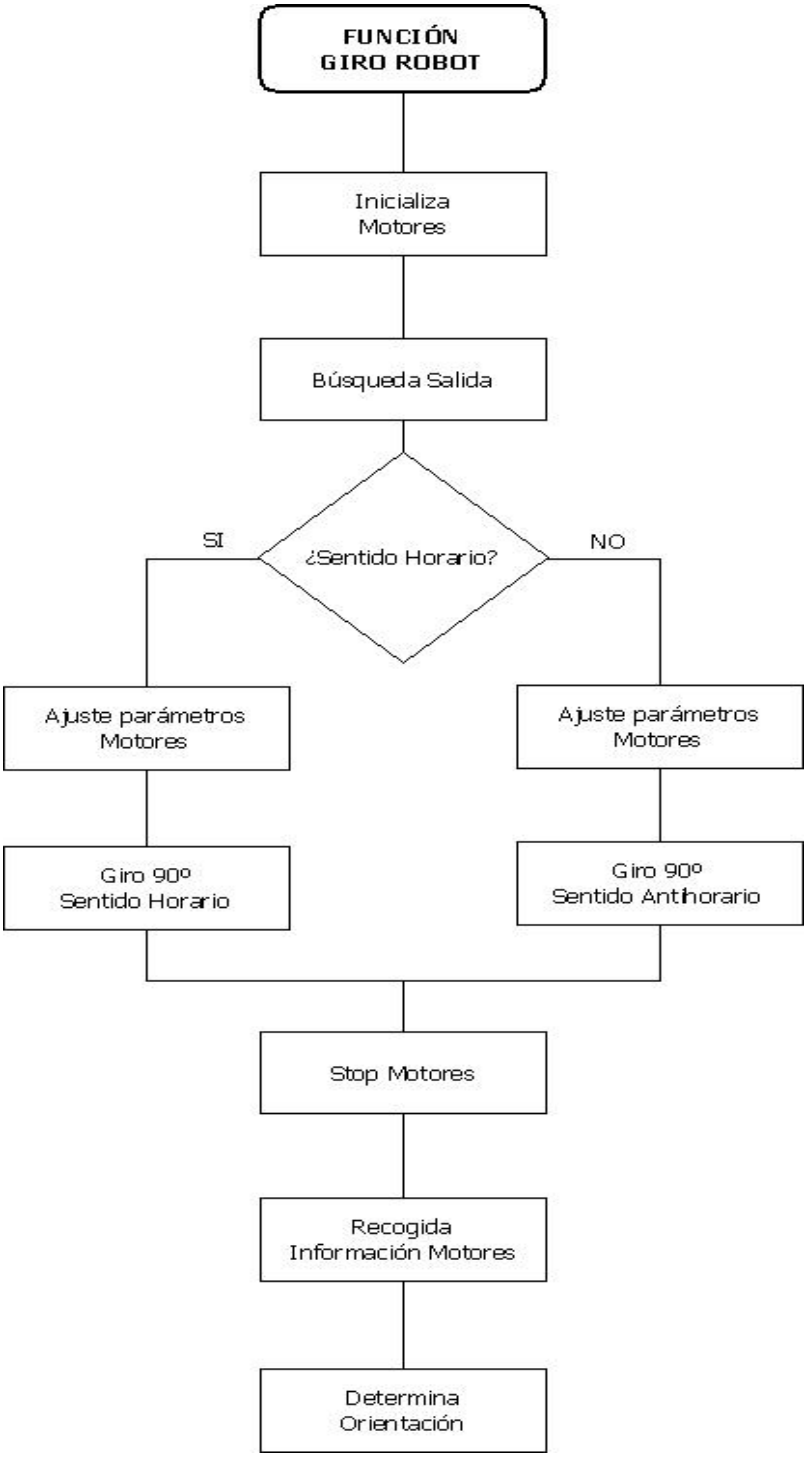


Figura 5.17: Diagrama de flujo de la subrutina Giro Robot

6. RESULTADOS EXPERIMENTALES

En este capítulo se presentan los resultados obtenidos durante la navegación del robot móvil por el entorno. Aunque a través de sus sensores es capaz de moverse por cualquier medio, en este caso, al tener que ejecutarse el algoritmo de localización en diferentes puntos del recorrido, es necesario disponer del mapa del entorno y sus medidas.

En principio, se parte de una localización absoluta, pasando posteriormente a realizar localizaciones locales durante la trayectoria del robot.

6.1. MAPA DEL ENTORNO

En el presente trabajo, la técnica utilizada para representar el entorno es el modelado de rejilla. Este sistema se basa en descomponer el universo en un conjunto de celdas, las cuáles pueden tomar dos valores, ocupadas o libres.

Las celdillas con fondo negro representan las celdillas ocupadas, es decir, son aquellas celdillas en las cuáles se encuentra algún obstáculo. Por otro lado, las celdillas con fondo blanco representan aquellas celdas que están libres, no hay presencia de obstáculos y, en consecuencia, serán las celdillas por donde se desplace el robot.

Las principales ventajas del modelado en rejilla son: su versatilidad, la posibilidad de trabajar con obstáculos de formas complicadas, la analogía con un mapa ordinario y el que, además, permite integrar de forma eficaz la lectura de varios sensores en el espacio y en el tiempo.

Su mayor desventaja estriba en que si el número de celdas del mapa no es lo suficientemente elevado, la calidad de la representación no es buena, pero, en caso contrario, el volumen de datos con el que se trabaja crece enormemente.

Los mapas, en principio, carecen de unidades reales, por lo que sus medidas están dadas en unidades de celda. Independientemente del mapa utilizado, el tamaño de la celda es el mismo, por tanto, el tamaño del mapa aumenta si se aumenta el número de celdas.

A continuación, se muestra la técnica de modelado de rejilla mediante la que se ha dividido el espacio.

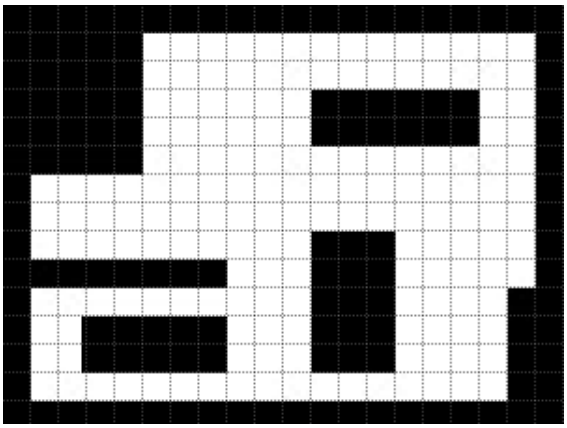


Figura 6.1: Representación de mapa de 20 x 15 celdas

En el siguiente ejemplo, podemos contemplar el incremento proporcional del tamaño del mapa en función del incremento del número de celdillas, tal y como comentamos anteriormente.

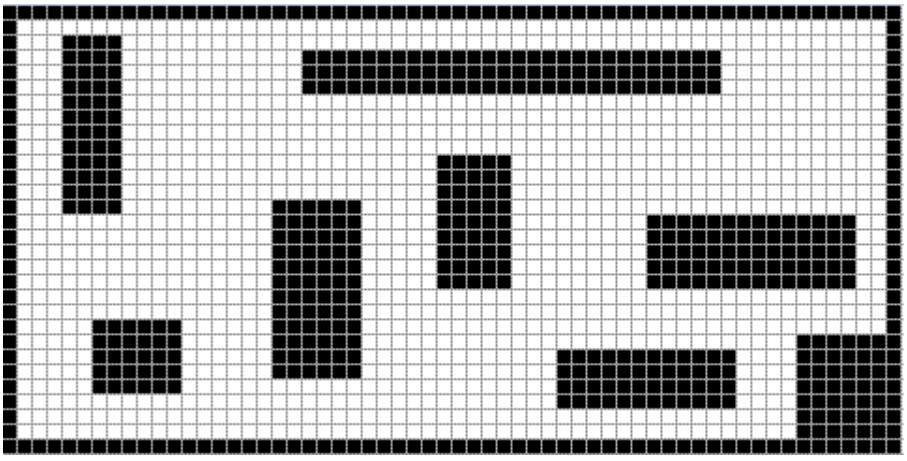


Figura 6.2: Representación de mapa de 60 x 30 celdas

Por último, nos centramos en el modelo de rejilla desarrollado para el presente proyecto. El diseño se corresponde con una representación a escala del laberinto por el que se va a mover el robot. Las dimensiones reales del mapa del entorno son de 142 cm de ancho por 182 cm de largo, sin embargo, los centímetros no son unidades válidas en la ejecución del algoritmo de localización, por lo que es necesario realizar una conversión, siendo, en nuestro caso, de una celda por cada centímetro de mapa, por lo que finalmente nos queda un mapa de rejilla de 142 celdas de ancho por 182 celdas de largo.

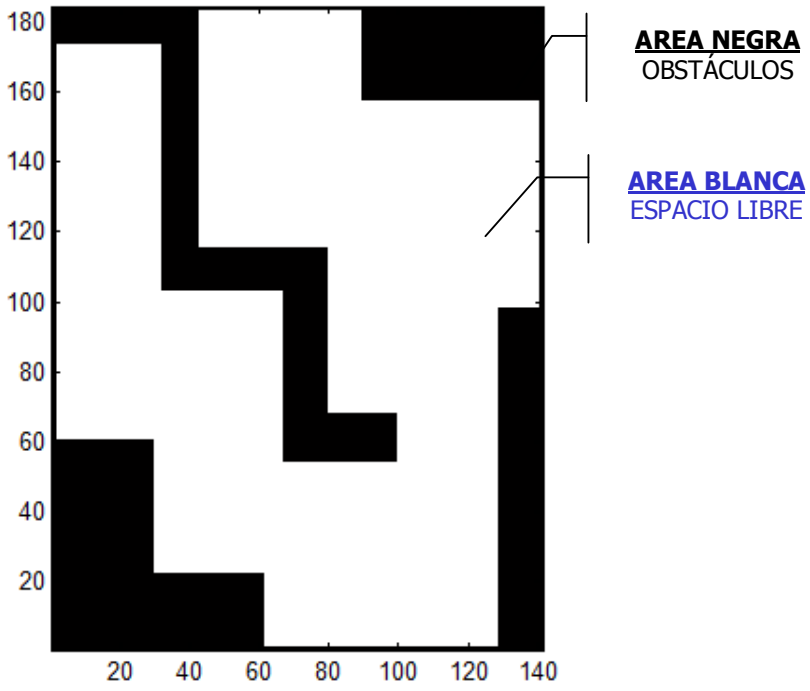


Figura 6.3: Mapa del entorno 142 x 182 celdas

6.2. CONFIGURACIÓN DE LOS PARÁMETROS DEL ALGORITMO DE EVOLUCIÓN DIFERENCIAL

Uno de los aspectos más complejos en el uso de algoritmos de evolución diferencial, es escoger la configuración de los parámetros, como pueden ser el tamaño de la población o las probabilidades de mutación y cruce.

Se trata de un hito desafiante, puesto que estos parámetros interactúan entre sí y tienen un gran efecto en el resultado de la búsqueda. Por este motivo, resulta imposible ajustarlos de forma independiente.

En el apartado 3.5.2., se detallan ciertos patrones para configurar los diferentes parámetros del algoritmo.

6.2.1. NÚMERO DE CROMOSOMAS

En estos algoritmos se considera que el cromosoma es toda la información del individuo codificada. Por tanto, (x,y,θ) es un cromosoma.

6.2.2. TAMAÑO DE LA POBLACIÓN

El tamaño de la población, N, indica el número de genes, o individuos, que forman una población.

Si la población es pequeña, es decir, si hay pocos individuos, es muy probable que el algoritmo tenga bajas probabilidades de dar un buen resultado, ya que sólo se tiene en consideración una pequeña parte del espacio de búsqueda.

Por el contrario, si la población es grande, con un número excesivo de individuos, se incrementaría la cantidad de variación presente en la población inicial a expensas de un mayor número de evaluaciones de la función de salud, que provocaría una ejecución del algoritmo muy lenta.

En el presente proyecto, se ha considerado una población de 250 individuos, resulta óptima para obtener un tiempo de cómputo no excesivamente elevado y suficientemente representativo para que la población inicial esté dispersa por todo el espacio de búsqueda.

6.2.3. MECANISMOS DE MUTACIÓN

La mutación se emplea para dar nueva información a la población y prevenir que la población se sature con cromosomas similares y se llegue a un problema de convergencia prematura.

6.2.3.1. ESTRATEGIA DE MUTACIÓN

El objetivo que buscamos es el de encontrar la estrategia que mejor se ajuste al problema. Existen 17 estrategias de mutación diferentes, basadas en las estrategias definidas por Storn y Price [36], que se han agrupado de la siguiente forma.

Las cuatro primeras estrategias utilizan un único vector de mutación (evolución de primer orden). Las trece estrategias restantes son de segundo orden, ya que utilizan dos vectores

diferentes para la mutación y se han agrupado según la "semilla" a partir de la cual se muta:

- **DE/1**

$$\vec{a}_i' = \vec{a}_{r3} + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 1)}$$

$$\vec{a}_i' = \vec{a}_i + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 2)}$$

$$\vec{a}_i' = \vec{a}_{best} + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 3)}$$

$$\vec{a}_i' = \vec{a}_{best\_old} + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 4)}$$

- **DE/2 Población Actual**

$$\vec{a}_i' = \vec{a}_i + F(\vec{a}_{best} - \vec{a}_i) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 5)}$$

$$\vec{a}_i' = \vec{a}_i + F(\vec{a}_{r3} - \vec{a}_i) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 6)}$$

$$\vec{a}_i' = \vec{a}_i + F(\vec{a}_{r1} - \vec{a}_{best}) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 7)}$$

- **DE/2 Mejor Población**

$$\vec{a}_i' = \vec{a}_{best} + F(\vec{a}_i - \vec{a}_{best}) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 8)}$$

$$\vec{a}_i' = \vec{a}_{best} + F(\vec{a}_{r3} - \vec{a}_{best}) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 9)}$$

$$\vec{a}_i' = \vec{a}_{best} + F(\vec{a}_{r1} - \vec{a}_{best}) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 10)}$$

$$\vec{a}_i' = \vec{a}_{best} + F\left(\vec{a}_{r1} - \vec{a}_{best\_old}\right) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 11)}$$

$$\vec{a}_i' = \vec{a}_{best} + F\left(\vec{a}_{r3} - \vec{a}_{best\_old}\right) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 12)}$$

$$\vec{a}_i' = \vec{a}_{best} + F\left(\vec{a}_{best\_old} - \vec{a}_{best}\right) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 13)}$$

- **DE/2 Mejor Población de la Iteración Anterior**

$$\vec{a}_i' = \vec{a}_{best\_old} + F\left(\vec{a}_{r1} - \vec{a}_{best\_old}\right) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 14)}$$

$$\vec{a}_i' = \vec{a}_{best\_old} + F\left(\vec{a}_{r3} - \vec{a}_{best\_old}\right) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 15)}$$

$$\vec{a}_i' = \vec{a}_{best\_old} + F\left(\vec{a}_{best} - \vec{a}_{best\_old}\right) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 16)}$$

$$\vec{a}_i' = \vec{a}_{best\_old} + F\left(\vec{a}_i - \vec{a}_{best\_old}\right) + F(\vec{a}_{r1} - \vec{a}_{r2}) \quad \text{(Estrategia 17)}$$

Tras realizar numerosas comprobaciones, se elige la estrategia 1 para el presente proyecto.

6.2.3.2. FACTOR DE MUTACIÓN F

El factor de mutación F es un parámetro del algoritmo que afecta a la variación diferencial. Se trata de una constante real comprendida entre 0 y 2.

Un alto factor de mutación aumenta la diversidad de población, por lo que se incrementa la capacidad del algoritmo para realizar una búsqueda global.

Una vez decidida una estrategia de mutación, se ha de elegir un valor adecuado de F. Para nuestro caso, se concluyó que el mejor valor para el factor de mutación con la estrategia de mutación 1, es F = 0.5.

6.2.4. MECANISMO DE CRUCE

La constante de cruce, indica con qué frecuencia se realiza la operación de cruce. El cruce se realiza esperando que los nuevos cromosomas posean las mejores características de los cromosomas viejos y que, incluso, sean superiores.

En el algoritmo se realiza un cruce de tipo binomial, ya que, como se expuso en el apartado 3.5.2.3., Price afirmaba que éste nunca será peor que un cruce de tipo exponencial. El cruce binomial responde a la siguiente expresión:

$$\vec{a}_i'' = \begin{cases} \vec{a}_i' \ si\ rand(i) \leq CR \\ \vec{a}_i \ si\ rand(i) > CR \end{cases} \quad (6.1)$$

Donde $\overrightarrow{a_i}'$ es la población obtenida a partir de la mutación y $\overrightarrow{a_i}$ es la población progenitora.

La constante de cruce está comprendida entre 0 y 1. Cuanto mayor es este parámetro, menos se parecerán los hijos a la generación precedente y más a los vectores procedentes de la mutación.

Para el presente proyecto, se ha estipulado que el valor óptimo para esta variable es de CR = 0.7.

6.2.5. MECANISMO DE SELECCIÓN

El operador selección compara la función salud de los individuos obtenidos tras la mutación y el cruce con la salud de los individuos originales.

Si el individuo modificado presenta mejor salud pasa a formar parte de la nueva generación, en otro caso, el individuo modificado se desestima.

6.2.6. CRITERIO DE PARADA

El criterio de parada se ha definido como el número de iteraciones que ha de realizar el algoritmo para encontrar un resultado satisfactorio.

Este número de iteraciones debe tener relación directa con el tamaño de la población elegida y, en consecuencia, con el tamaño del mapa del entorno.

Consideramos un número de iteraciones suficiente para obtener un buen resultado al número entero inmediatamente superior al de la siguiente ecuación:

$$Iteraciones = 0.0028 \times N^2 + 12.184 \qquad (6.2)$$

Donde N representa la población, por lo que para nuestro caso, el número de iteraciones sería de 188 (por sencillez utilizaremos 190 iteraciones).

6.3. EJECUCIÓN DEL PROGRAMA

Cómo ya se ha comentado en apartados anteriores, el presente proyecto persigue la consecución de dos objetivos.

El robot móvil utilizado, Lego Mindstorms NXT, a través de su sensor de ultrasonido, será capaz de tomar decisiones que le permitan la planificación de estrategias de movimiento con el fin de esquivar cualquier obstáculo que se encuentre en su trayectoria, con el fin de encontrar la salida del laberinto.

De forma simultánea, cada vez que el robot se encuentre detenido frente a un obstáculo, se lanzará la ejecución de un algoritmo de evolución diferencial que le permitirá al robot relocalizarse dentro de un mapa de entorno conocido, proporcionándonos datos relativos a su posición y orientación.

En este apartado, se trata de explicar secuencialmente y de la forma más fiel posible, cuál es el proceso aplicado para alcanzar sendos objetivos.

Los datos que se presentan a continuación, corresponden a una ejecución del programa. Hay que tener en cuenta que los resultados obtenidos son específicos de cada ejecución del programa realizado. Básicamente, esto se debe a que el algoritmo de localización es un proceso estocástico y, además, a que las medidas proporcionadas por los sensores internos y externos varían entre las ejecuciones.

6.3.1. SEÑAL DE SALIDA

En el diseño del robot se ha incluido un sensor de sonido, cuyas características están descritas en el apartado 4.2.3.2., el cuál se habilitará en el comienzo de la ejecución del programa principal.

El robot permanecerá inmóvil en la posición inicial, sin realizar ningún tipo de función hasta que dicho sensor no registre un sonido superior al 50% en su escala de medida, tabla 4.1., que se interpretará como la señal necesaria para continuar con la ejecución del resto del programa y la consecuente deshabilitación del sensor de sonido [Apéndice A].

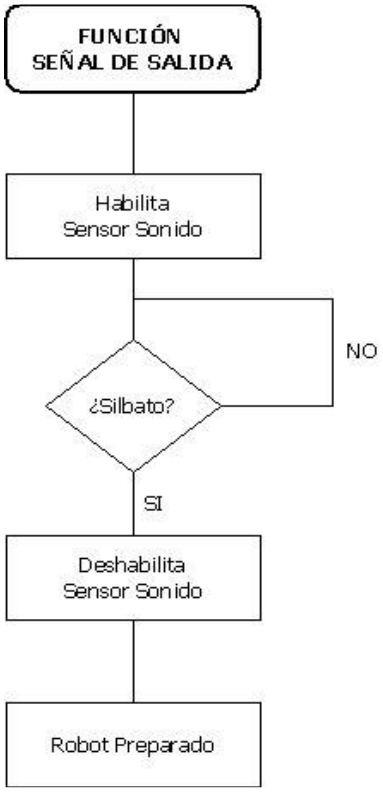


Figura 6.4: Diagrama de Flujo de la función señal de salida

Dicha "barrera" sonora se puede alcanzar de forma sencilla, sólo se precisará de una palmada relativamente cerca del sensor o de la utilización de algún elemento externo como puede ser un silbato.

Aparte de conseguir que la ejecución de la parte central del programa se lleve a cabo cuando nosotros deseamos, la inclusión de esta condición establecida por el sensor de sonido, nos permite hacernos una ligera idea de la versatilidad de las acciones que se pueden llevar a cabo con el robot Lego Mindstorms y sus distintos sensores.

6.3.2. LOCALIZACIÓN INICIAL

Cómo se ha explicado en capítulos anteriores, el método utilizado es una localización absoluta, esto significa que no tenemos ninguna información referente del punto inicial que ocupa nuestro robot.

En este escenario, el primer paso obligatorio a realizar es un chequeo utilizando el sensor ultrasonido, para extraer información del entorno que le rodea.

En las siguientes instantáneas se puede observar la posición ocupada por el sensor ultrasonido durante la rotación de éste.

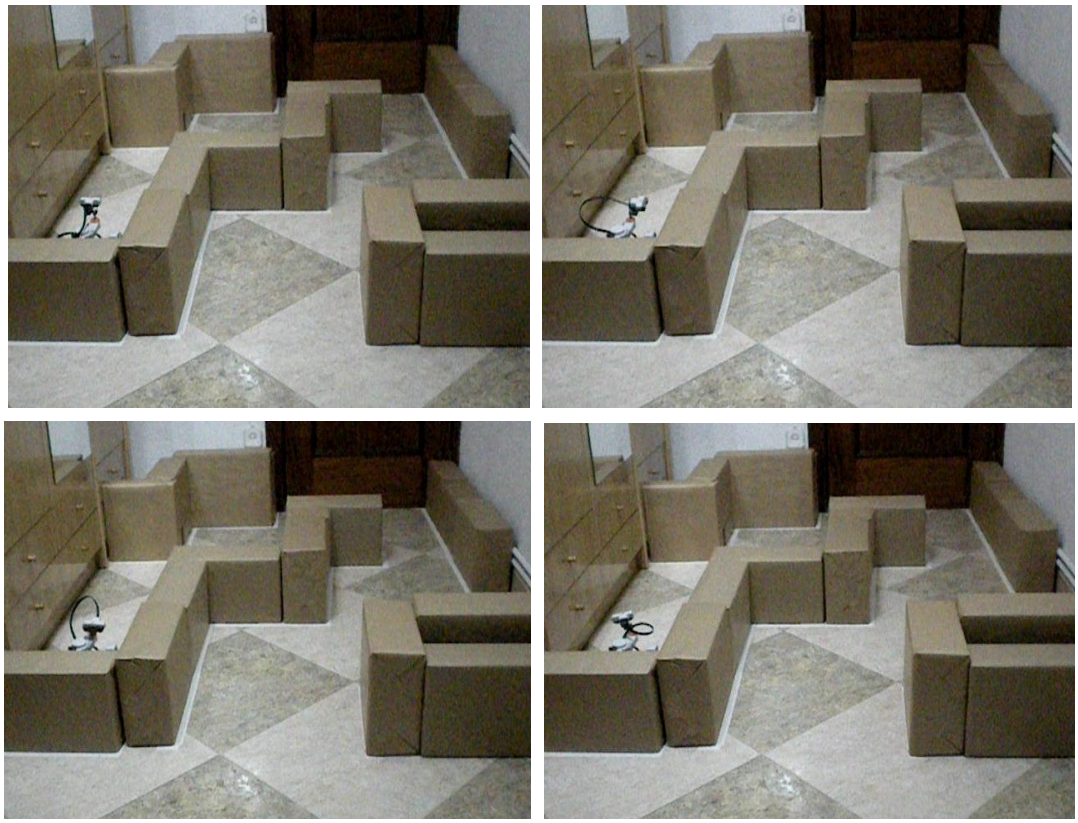


Figura 6.5: Arriba izquierda: Sensor ultrasonido en 0º. Arriba derecha: Sensor ultrasonido en 90º. Abajo izquierda: Sensor ultrasonido en 180º. Abajo derecha: Sensor ultrasonido en 330º.

Como resultado, obtenemos la representación del siguiente diagrama polar y, con ello, el vector distancia que resulta indispensable para la ejecución del algoritmo evolutivo diferencial, ya que nos proporciona la relación de distancia en función del ángulo ocupado en ese momento por el sensor ultrasonido del robot.

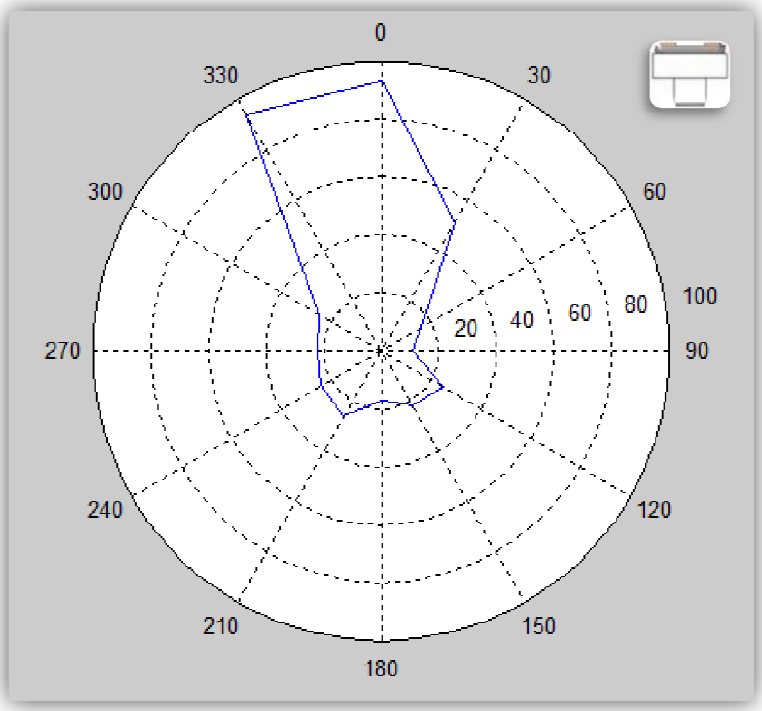


Figura 6.6: Diagrama polar posición inicial

Ángulo [º]	Distancia [cm]
0	93
30	51
60	16
90	11
120	25
150	22
180	17
210	26
240	24
270	22
300	25
330	94
360	93

Tabla 6.1: Relación ángulo-distancia posición inicial

Con el vector distancia proporcionado por el sensor de ultrasonido como única información de referencia, se pasa a ejecutar el algoritmo evolutivo diferencial que nos proporcionará una estimación de la posición y orientación que ocupa el robot.

6.3.2.1. EVOLUCIÓN DE LA POBLACIÓN

En condiciones normales, cuando se ejecuta el algoritmo de localización, no vemos el proceso interno realizado en éste para la obtención del resultado final, que será mostrado por pantalla. Sin embargo, consideramos fundamental explicar qué sucede en el camino para comprender de forma clara el funcionamiento del algoritmo.

Por ese motivo, a lo largo de este capítulo, se mostrarán los pasos intermedios que tienen lugar en el algoritmo de localización. Donde se partirá de una población inicial en torno a un área, la cual se irá reduciendo a medida que la población se vea sometida a las operaciones de cruce y mutación hasta, finalmente, converger al resultado final.

Para este primer caso, se puede observar que en las primeras iteraciones de la ejecución del algoritmo, la población inicial está distribuida estocásticamente por todo el área libre del mapa de entorno. Esto es así, debido a que la información con la que cuenta el algoritmo es muy limitada (vector distancia) y al número bajo de iteraciones realizadas hasta el momento.

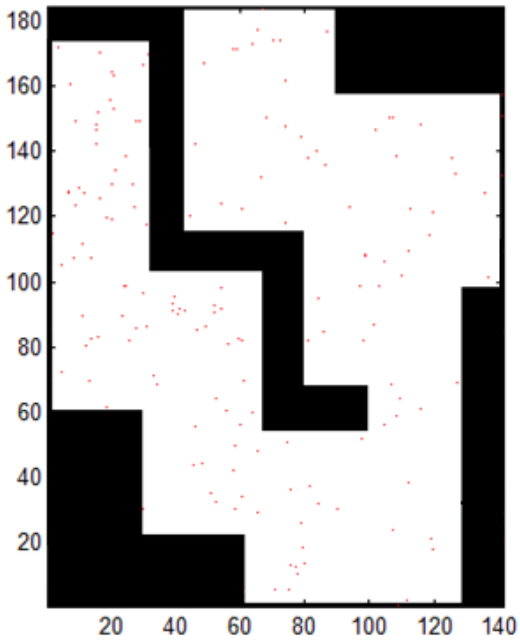


Figura 6.7: Evolución de la población con 10 iteraciones
Mejor individuo: [x = 19.18047, y = 154.09837, θ = 358.79137]

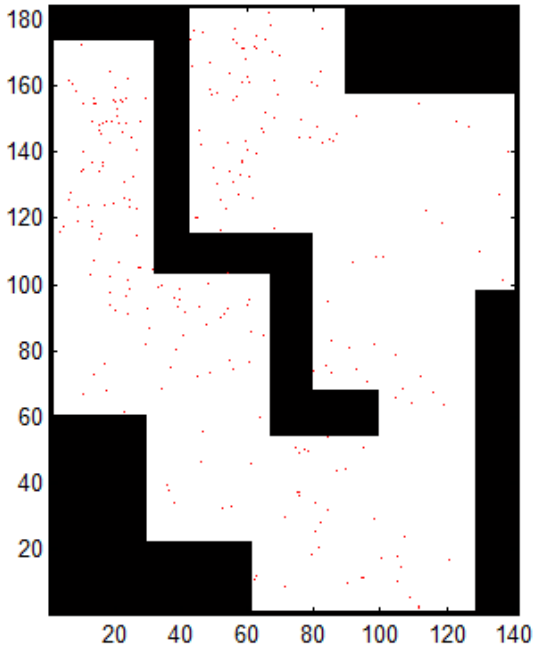


Figura 6.8: Evolución de la población con 50 iteraciones
Mejor individuo: [x = 19.32066, y = 155.72127, θ = 5.01431]

Según se incrementa el número de iteraciones realizadas por el algoritmo evolutivo diferencial, se puede observar cómo la población existente acusa una disgregación menor. Esto se debe a que la población se va concentrando paulatinamente en torno a los puntos que poseen mayor salud, es decir, aquéllos que presentan mayores probabilidades de ser el resultado final.

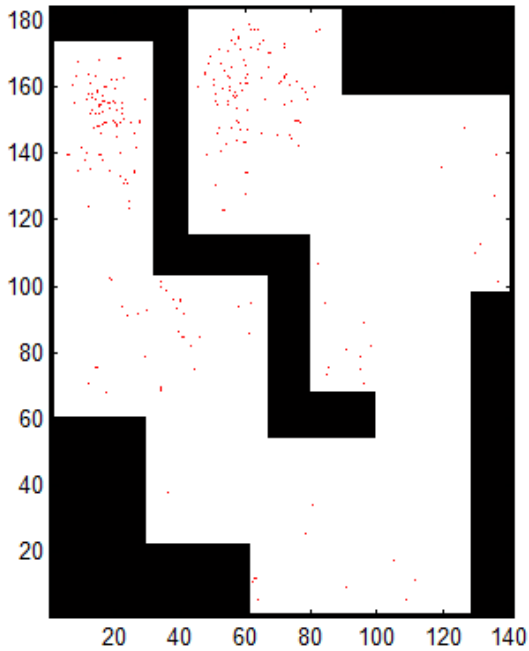


Figura 6.9: Evolución de la población con 100 iteraciones
Mejor individuo: [x = 18.86214, y = 154.15274, θ = 4.86393]

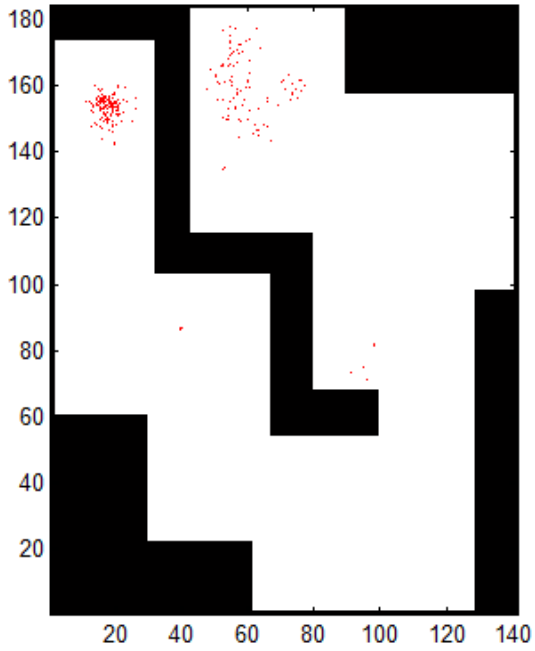


Figura 6.10: Evolución de la población con 160 iteraciones
Mejor individuo: [x = 19.94878, y = 154.49373, θ = 358.52228]

Al final de la ejecución del algoritmo, una cruz de color azul representa gráficamente el punto estimado de la posición que ocupa nuestro robot. Este punto es el que presenta mayor salud dentro del mapa del entorno.

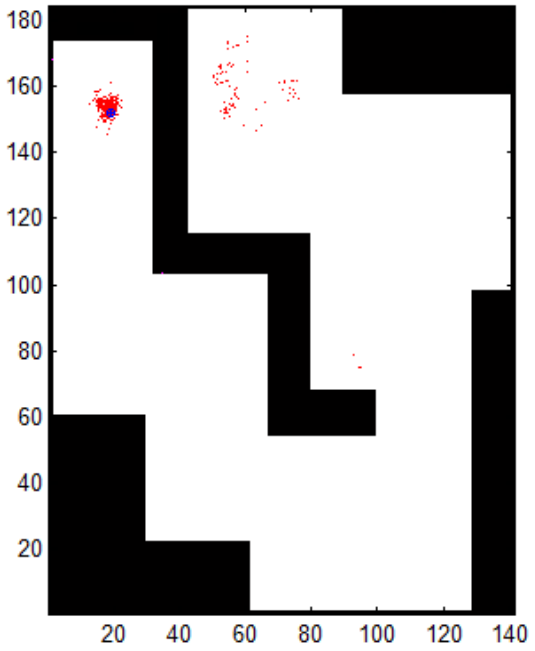


Figura 6.11: Evolución de la población con 200 iteraciones
Mejor individuo: [x = 19.06214, y = 152.21478, θ = 358.07700]

6.3.3. PRIMERA SECUENCIA

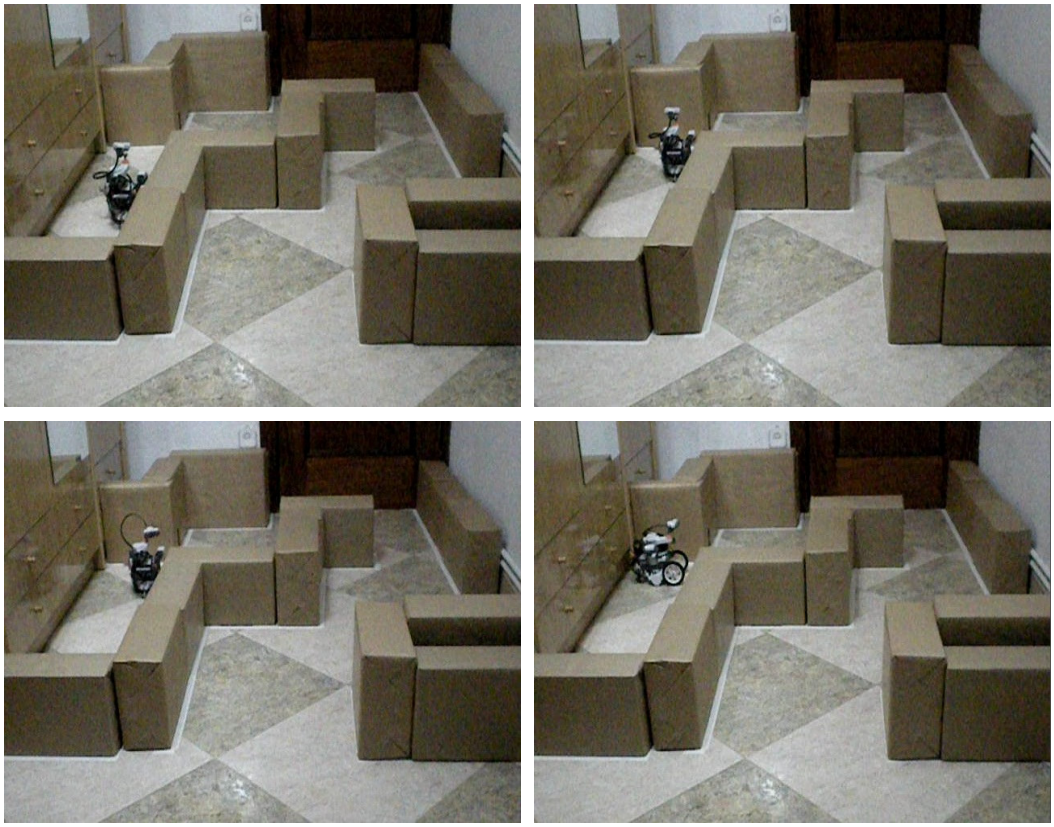


Figura 6.12: Arriba izquierda: Robot móvil avanza. Arriba derecha: Robot móvil se detiene. Abajo izquierda: Robot móvil realiza chequeo del entorno. Abajo derecha: Robot móvil gira.

Una vez que el robot queda localizado inicialmente, éste comienza a avanzar con una velocidad constante y con el sensor ultrasonido habilitado con el objetivo de registrar de forma continua la presencia de obstáculos.

En el momento que el sensor registra un obstáculo a menos de 25 cm de distancia, se manda de forma inmediata la orden de parar los motores para evitar que el robot colisione con el objeto, en nuestro caso, los límites del laberinto.

Al desplazarse, el robot ocupa una nueva posición dentro del entorno, por lo que tendrá que volver a localizarse. Para ello realizará un chequeo de 360º grados con el sensor ultrasonido para obtener el vector distancia. Además, cuenta con la información proporcionada por los sensores odométricos alojados en el motor de las ruedas motrices que, para este caso, corresponde a 77,14 cm. El diagrama polar obtenido es el que se muestra en la siguiente figura.

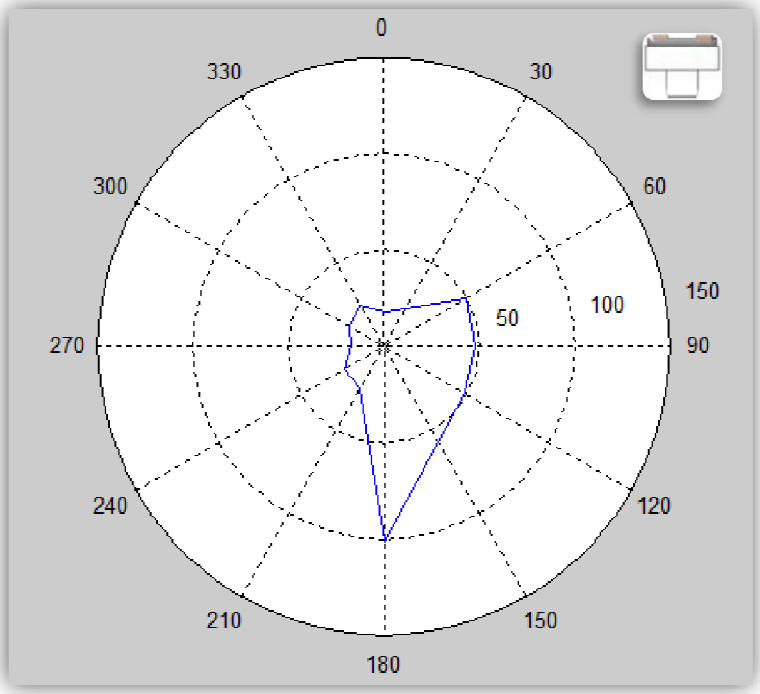


Figura 6.14: Diagrama polar posición 1

Ángulo [º]	Distancia [cm]
0	18
30	23
60	50
90	48
120	49
150	57
180	101
210	25
240	24
270	17
300	21
330	25
360	18

Tabla 6.2: Relación ángulo-distancia posición 1

En los resultados de la tabla, se puede observar que la distancia máxima registrada es la correspondiente a 180º, pero esto implicaría el retroceso del robot, que no es una situación deseable, por lo que como

se detalla en el apartado 5.3.3. del presente proyecto, las medidas correspondientes a esta situación, de 150º a 210º, serán descartadas.

Teniendo en cuenta esa condición, la mayor distancia registrada es la correspondiente a 60º, por lo tanto, la subrutina giro se ejecutará y, atendiendo a lo explicado en el capítulo 5, el robot girará 90º en sentido horario.

Con estos datos, el vector distancia y la información relativa a la odometría, se vuelve a ejecutar el algoritmo de localización.

6.3.3.1. EVOLUCIÓN DE LA POBLACIÓN

Al contrario que en la primera localización, se puede observar cómo, desde la primera iteración, la población, principalmente, se concentra en torno a un área determinada dentro del mapa del entorno.

Esencialmente, esto es debido a que, en esta ocasión, el algoritmo evolutivo diferencial cuenta con más información que en el caso anterior, como son los datos relativos a la posición y orientación de origen, la distancia recorrida en el avance según los sensores odométricos y el vector distancia de la posición actual.

La evolución de la población hasta obtener el resultado final se muestra en las siguientes figuras:

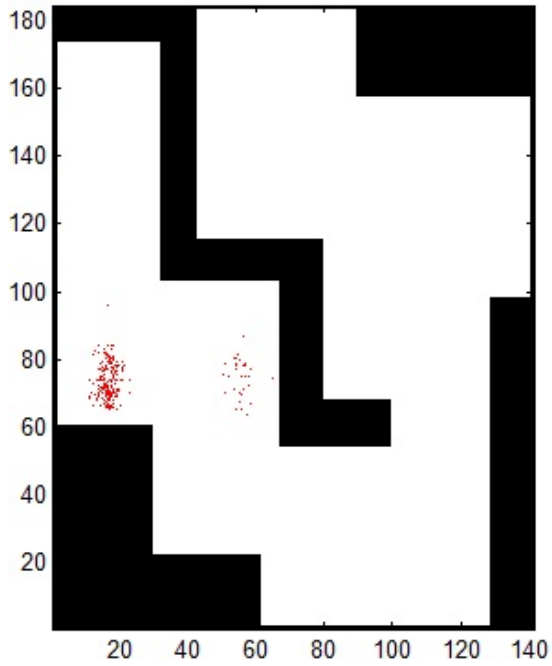


Figura 6.13: Evolución de la población con 10 iteraciones
Mejor individuo: [x = 14.01435, y = 78.76245, θ = 4,42853]

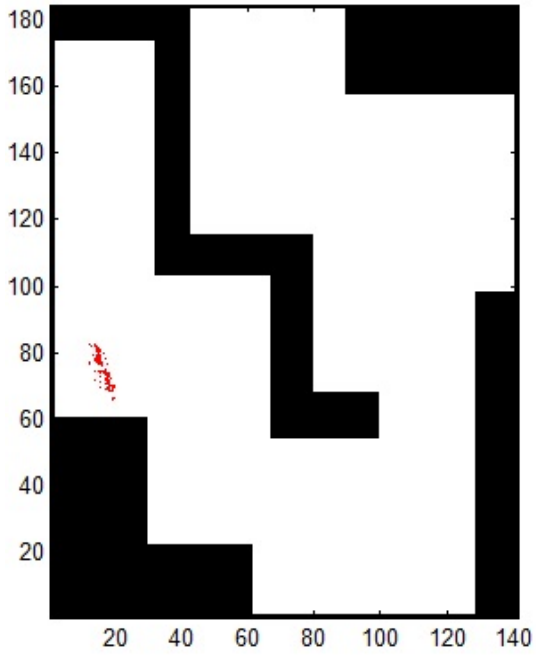


Figura 6.14: Evolución de la población con 50 iteraciones
Mejor individuo: [x = 15.56476, y = 76.09366, θ = 4.54333]

Podemos observar que, a medida que aumenta el número de iteraciones realizadas por el algoritmo, el área que representan las soluciones con mayor probabilidad de ser el resultado final (mayor salud) se concretan en torno a un área muy concreta.

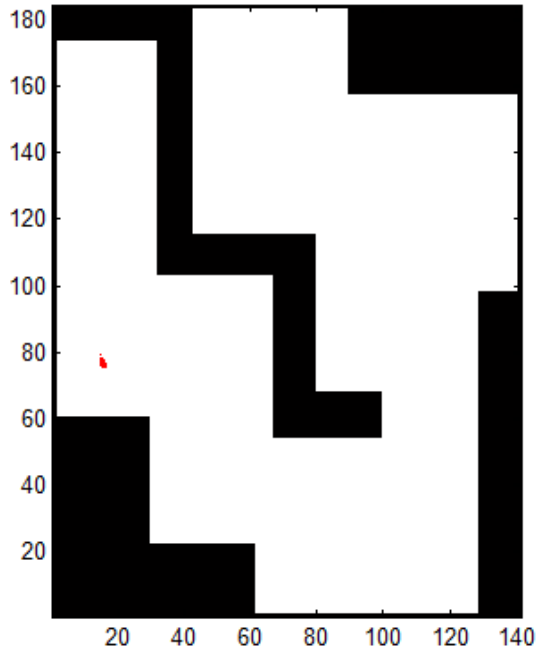


Figura 6.15: Evolución de la población con 100 iteraciones
Mejor individuo: [x = 15.54518, y = 76.51875, θ = 4.77337]

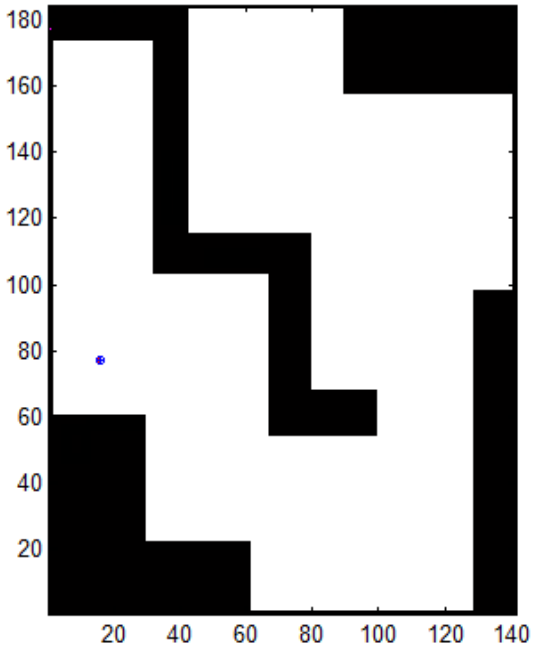


Figura 6.16: Evolución de la población con 130 iteraciones
Mejor individuo: [x = 15.62062, y = 77.41566, θ = 4.34156]

El gráfico anterior nos muestra el resultado final con la posición ocupada por el robot. Haciendo uso de una figura con la zona en cuestión ampliada, se puede observar en detalle cómo en torno al punto final (cruz azul) existe una alta concentración de la población (puntos rojos).

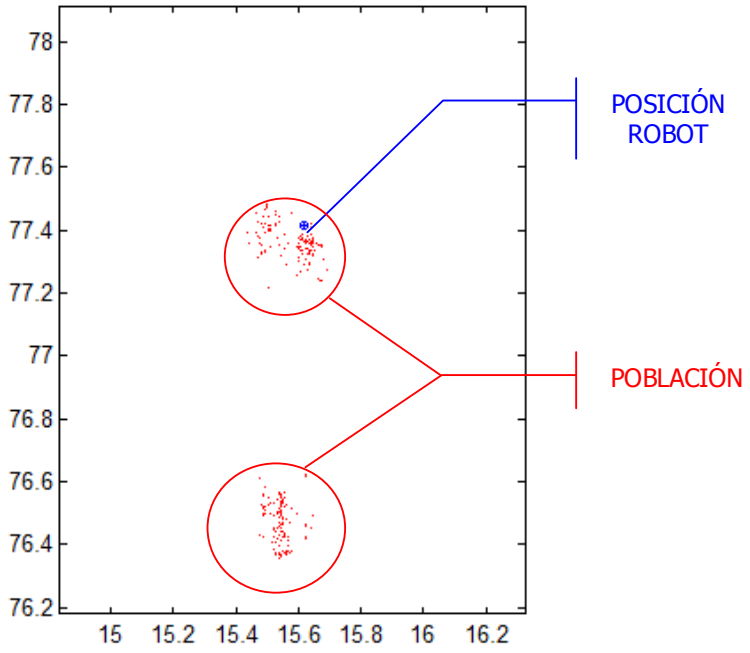


Figura 6.17: Detalle de la población con 130 iteraciones
Mejor individuo: [x = 15.62062, y = 77.41566, θ = 4.34156]

6.3.4. SEGUNDA SECUENCIA

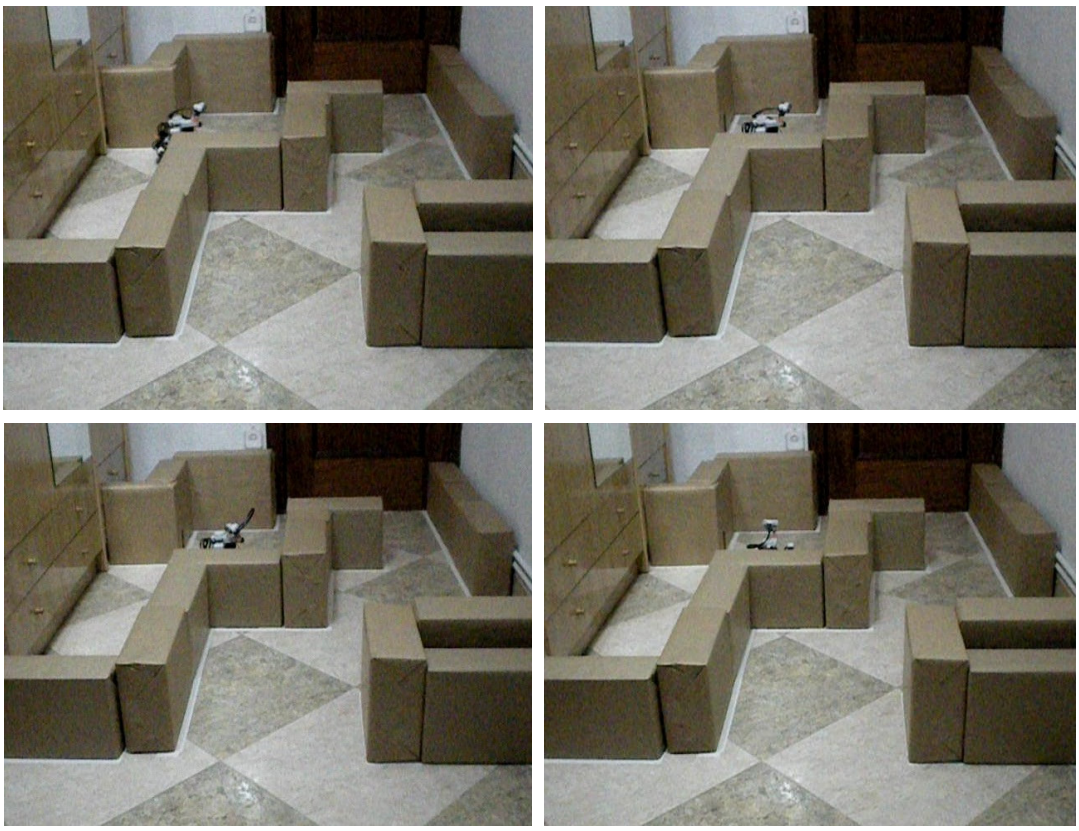


Figura 6.18: Arriba izquierda: Robot móvil avanza. Arriba derecha: Robot móvil se detiene. Abajo izquierda: Robot móvil realiza chequeo del entorno. Abajo derecha: Robot móvil gira.

El robot reinicia su marcha y avanzará hasta que el sensor ultrasonido detecte la presencia de un obstáculo, de forma análoga a lo anteriormente descrito.

Inmediatamente después de pararse, el sensor ultrasonido comenzará a realizar la toma de medidas a su alrededor para la obtención del vector distancia.

En esta ocasión la información odométrica obtenida es de a 32,62 cm y el diagrama polar resultante se muestra en la figura 6.19.

Se observa que, la distancia mayor obtenida es de 58 cm, que corresponde a 240º. Al encontrarse en el semiplano izquierdo, el robot ejecutará la orden de giro de 90º en sentido antihorario y, posteriormente, quedará dispuesto para ejecutar el algoritmo de localización

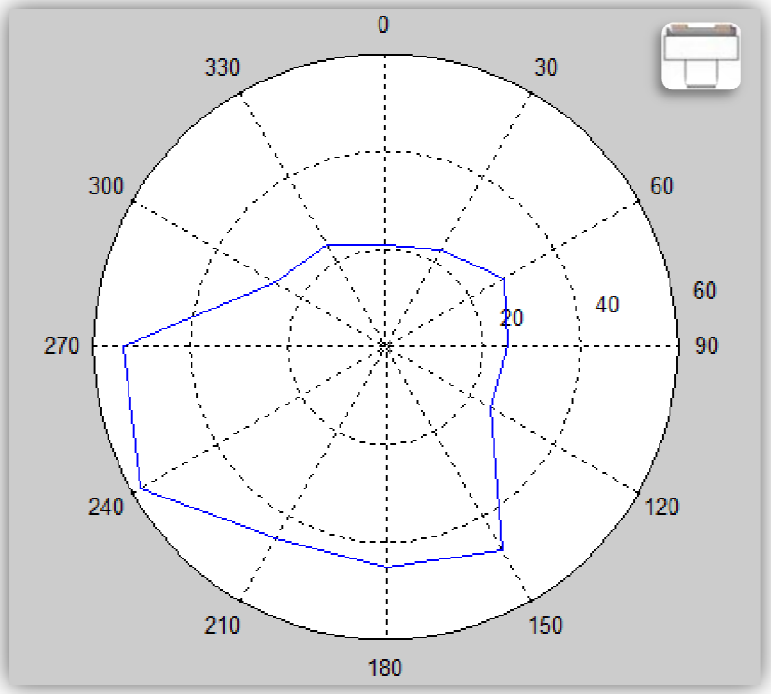


Figura 6.19: Diagrama polar posición 2

Ángulo [º]	Distancia [cm]
0	21
30	23
60	28
90	25
120	25
150	48
180	45
210	45
240	58
270	54
300	26
330	24
360	21

Tabla 6.3: Relación ángulo-distancia posición 2

Al contrario que en el diagrama polar correspondiente a la primera posición, en este caso, las medidas correspondientes a la media vuelta, de 150º a 210º, no resultan ser las de mayor valor, por lo que no sería necesario discriminarlas; sin embargo, por defecto, está programado para que estas medidas siempre sean obviadas.

6.3.4.1. EVOLUCIÓN DE LA POBLACIÓN

A continuación, se muestran diferentes gráficos referentes a la evolución de la población, para la nueva localización del robot. En ellos se pueden observar cómo el área ocupada por la población, desde las primeras iteraciones del algoritmo, es más reducido y acotado que para los dos casos anteriores.

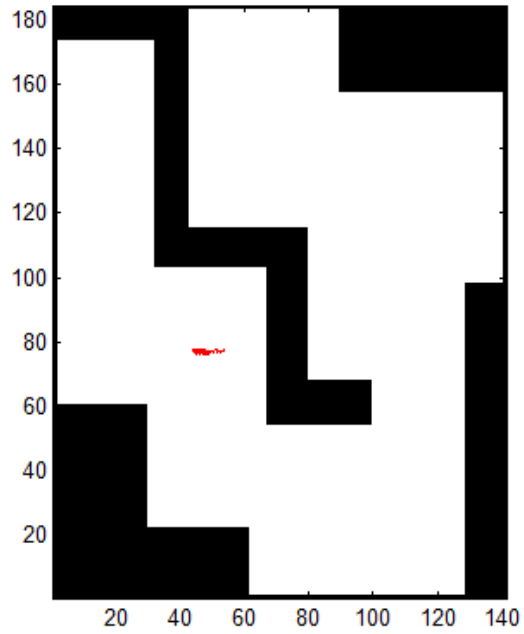


Figura 6.20: Evolución de la población con 10 iteraciones
Mejor individuo: [x = 50.60160, y = 77.46956, θ = 95.41322]

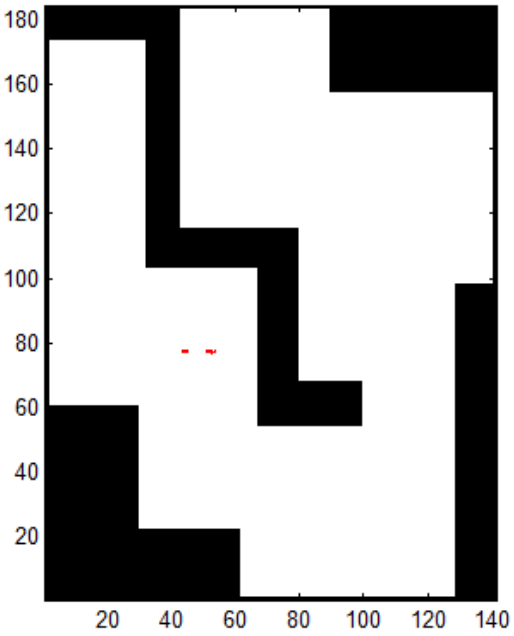


Figura 6.21: Evolución de la población con 30 iteraciones
Mejor individuo: [x = 51.05326, y = 77.74646, θ = 95.08309]

A partir de un cierto número de iteraciones, para este caso se encuentra en torno a las 80 iteraciones, vemos que el área ocupada por la población candidata a solución es mínima, por lo que es necesario hacer uso de una imagen ampliada para ser capaz de apreciar alguna diferencia, tal y como se observa en la figura 6.22.

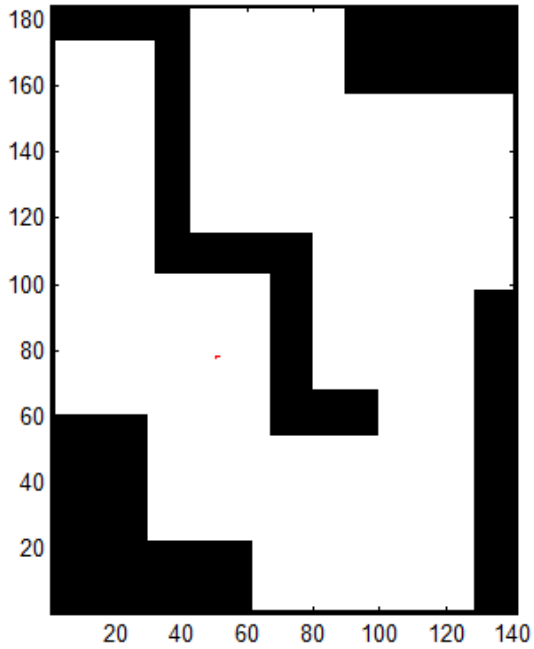


Figura 6.21: Evolución de la población con 80 iteraciones
Mejor individuo: [x = 50.48993, y = 78.42404, θ = 95.31819]

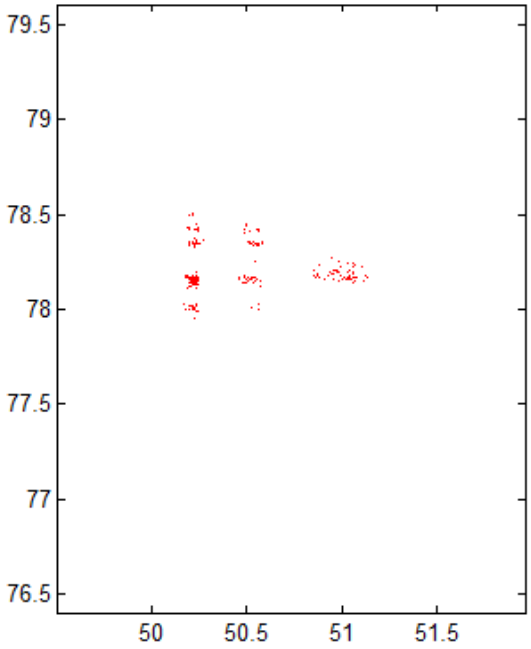


Figura 6.22: Detalle de la población con 80 iteraciones
Mejor individuo: [x = 50.48993, y = 78.42404, θ = 95.31819]

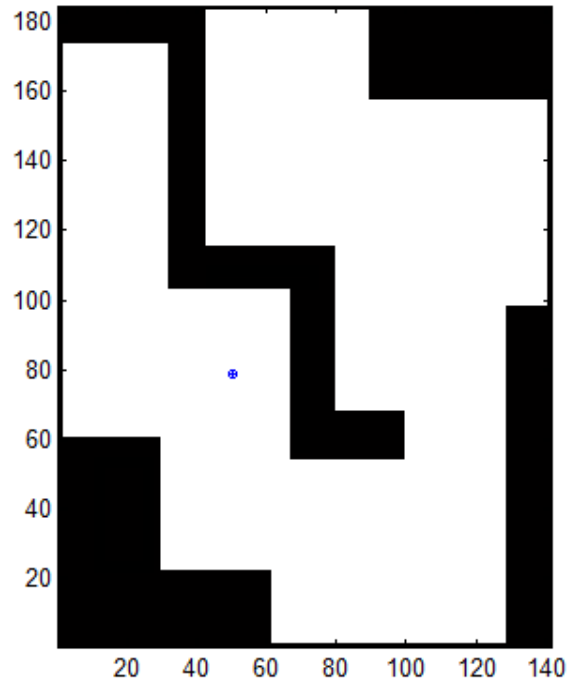


Figura 6.23: Evolución de la población con 130 iteraciones
Mejor individuo: [x = 50.21969, y = 78.81854, θ = 94.63744]

Finalmente, se muestra el resultado obtenido por el algoritmo. Podemos observar que para esta posición no ha sido necesario realizar las 190 iteraciones establecidas como criterio de parada al inicio de la ejecución del programa, ya que el algoritmo ha convergido al resultado cuando llevaba realizadas 130 iteraciones. Esto mismo también ocurría en la posición anterior (posición 1), donde al algoritmo también le bastó con 130 iteraciones.

6.3.5. TERCERA SECUENCIA

Figura 6.24: Arriba izquierda: Robot móvil avanza. Arriba derecha: Robot móvil se detiene. Abajo izquierda: Robot móvil realiza chequeo del entorno. Abajo derecha: Robot móvil gira.

Como el robot aún se encuentra en el laberinto, es necesario que siga repitiendo la secuencia descrita para los casos anteriores.

En primer lugar, el robot avanzará de nuevo hasta la detección del siguiente obstáculo, obteniendo los siguientes resultados odométricos: 34,51 cm.

Tras el chequeo de 360º realizado por el sensor ultrasonido, el diagrama polar obtenido se muestra en la siguiente figura:

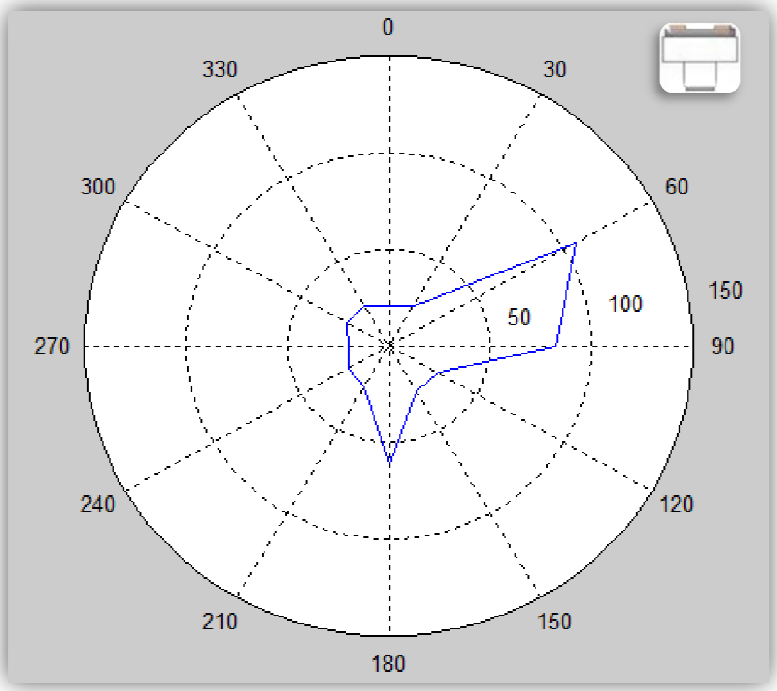


Figura 6.25: Diagrama polar posición 3

Ángulo [º]	Distancia [cm]
0	21
30	24
60	107
90	82
120	28
150	27
180	61
210	25
240	23
270	20
300	24
330	24
360	21

Tabla 6.3: Relación ángulo-distancia posición 3

En la anterior tabla, observamos que la distancia máxima se registra cuando el sensor de ultrasonido se encuentra en 60º con 107 cm. Esta medida corresponde al semiplano derecho, en consecuencia, el robot cambiará su orientación en 90º girando en sentido horario.

6.3.5.1. EVOLUCIÓN DE LA POBLACIÓN

A continuación, se muestra la evolución de la población durante la ejecución del algoritmo evolutivo diferencial que, como en el resto de casos, nos dará como resultado la localización del robot, precisando su posición y orientación dentro del mapa.

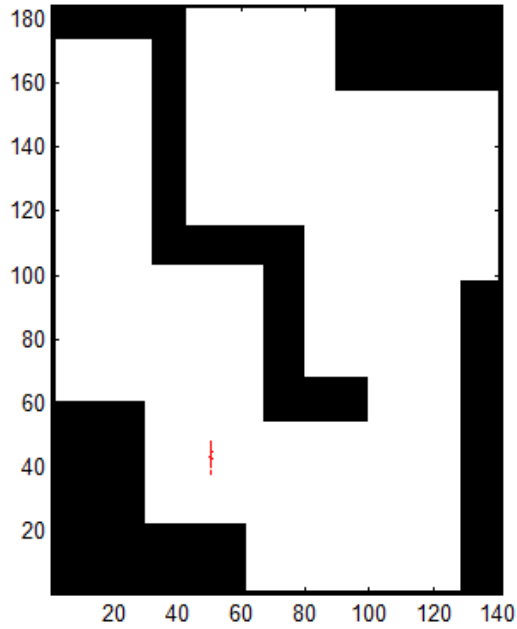


Figura 6.26: Evolución de la población con 10 iteraciones
Mejor individuo: [x = 50.21532, y = 38.44827, θ = 8.73141]

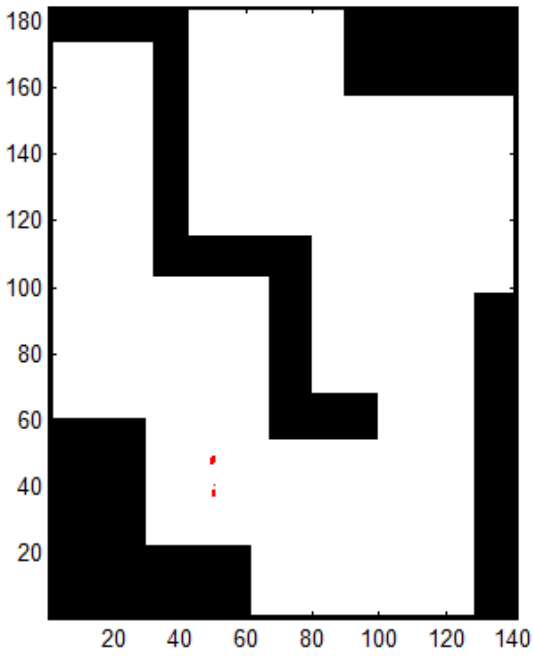


Figura 6.27: Evolución de la población con 50 iteraciones
Mejor individuo: [x = 50.21496, y = 37.65535, θ = 8.67893]

Observamos que, llegado a un número determinado de iteraciones, el área que comprende los posibles resultados es muy reducida y es necesario utilizar una ampliación de la zona en cuestión para poder observar con nitidez la evolución de la población.

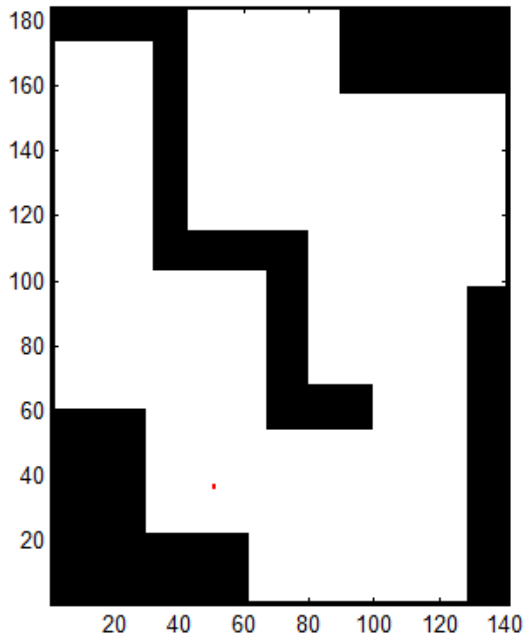


Figura 6.28: Evolución de la población con 100 iteraciones
Mejor individuo: [x = 50.80645, y = 36.84075, θ = 7.71817]

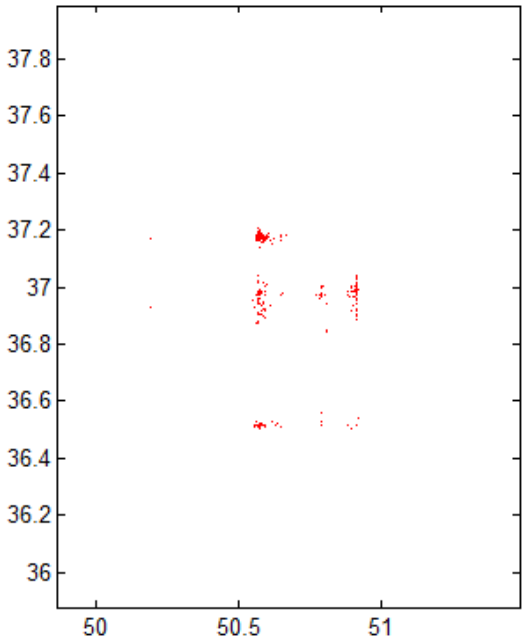


Figura 6.29: Detalle de la población con 100 iteraciones
Mejor individuo: [x = 50.80645, y = 36.84075, θ = 7.71817]

Finalmente, el algoritmo converge al resultado final cuando se han ejecutado 180 iteraciones.

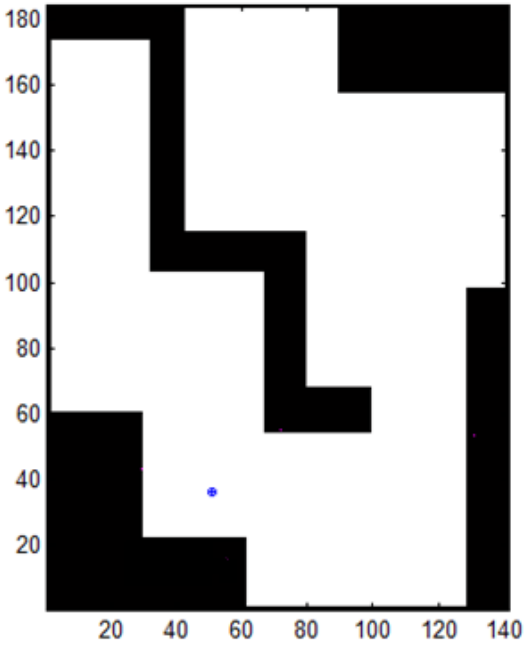


Figura 6.30: Detalle de la población con 180 iteraciones
Mejor individuo: [x = 50.94440, y = 36.22440, θ = 7.23021]

6.3.6. CUARTA SECUENCIA

Figura 6.32: Arriba izquierda: Robot móvil avanza. Arriba derecha: Robot móvil se detiene. Abajo izquierda: Robot móvil realiza chequeo del entorno. Abajo derecha: Robot móvil gira.

De forma análoga a lo anteriormente descrito, el robot activará los motores motrices para reiniciar su marcha, avanzando hasta que el sensor ultrasonido detecte la presencia de un obstáculo.

Tras el avance del robot, la distancia realizada por éste según los sensores odométricos es de: 68,22 cm.

En la siguiente representación se observa el diagrama polar obtenido tras el chequeo realizado por el robot. Se puede observar con claridad, tanto en la figura 6.3.1. como en la tabla 6.4., que la medida mayor corresponde a la de 114 cm que se registra a 90º, por tanto, al encontrarse en el semiplano derecho, el robot, como ya hiciera anteriormente, volverá a girar 90º en sentido horario.

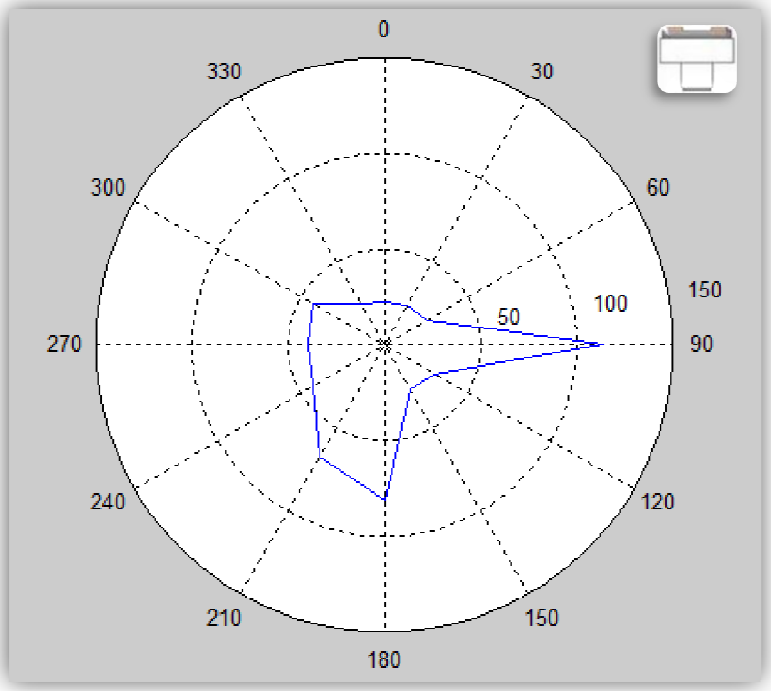


Figura 6.31: Diagrama polar posición 4

Ángulo [º]	Distancia [cm]
0	22
30	24
60	26
90	114
120	30
150	27
180	81
210	67
240	43
270	40
300	43
330	25
360	22

Tabla 6.4: Relación ángulo-distancia posición 4

6.3.6.1. EVOLUCIÓN DE LA POBLACIÓN

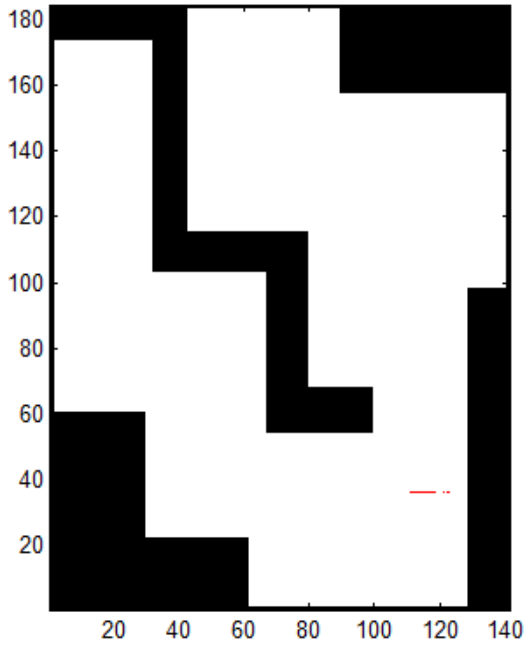


Figura 6.30: Detalle de la población con 10 iteraciones
Mejor individuo: [x = 112.47393, y = 36.27227, θ = 87.14437]

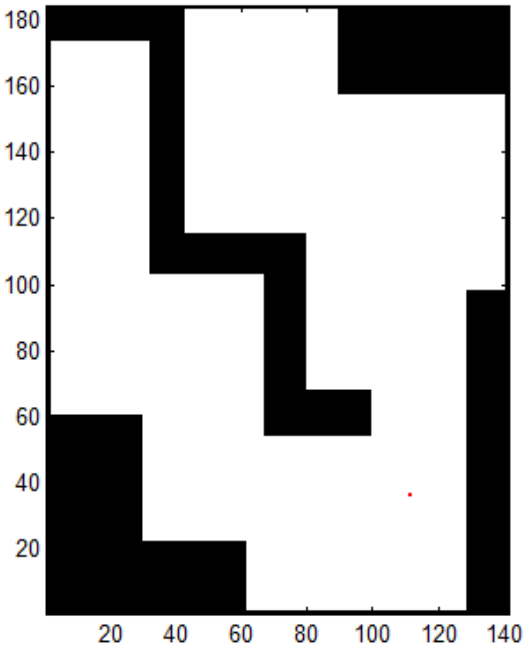


Figura 6.31: Detalle de la población con 40 iteraciones
Mejor individuo: [x = 111.30777, y = 37.04274, θ = 85.87403]

Al algoritmo evolutivo diferencial le basta con 70 iteraciones para converger al resultado final y mostrarlo tal y como se aprecia en la figura 6.32.

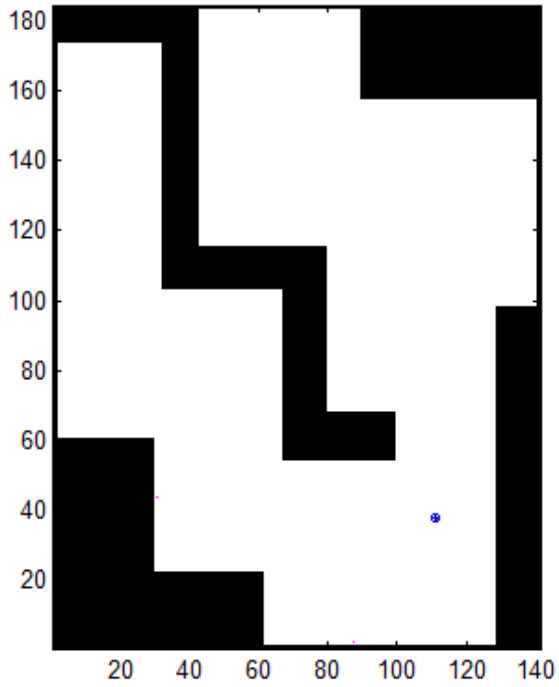


Figura 6.32: Detalle de la población con 70 iteraciones
Mejor individuo: [x = 111.28679, y = 38.03163, θ = 85.88018]

6.3.7. QUINTA SECUENCIA

Figura 6.33: Arriba izquierda: Robot móvil avanza. Arriba derecha: Robot móvil se detiene. Abajo izquierda: Robot móvil realiza chequeo del entorno. Abajo derecha: Robot móvil gira.

En esta secuencia, el robot avanzará de la posición 4 a la posición 5. Se puede observar en las imágenes de la parte superior, que la distancia a recorrer es la mayor de todas las existentes en el laberinto, siendo la distancia recorrida de 100,87 cm la información proporcionada por los sensores. Al ser mayor la distancia a cubrir, las probabilidades de que la odometría acumule errores o exista algún factor que influya en la medida también es mayor.

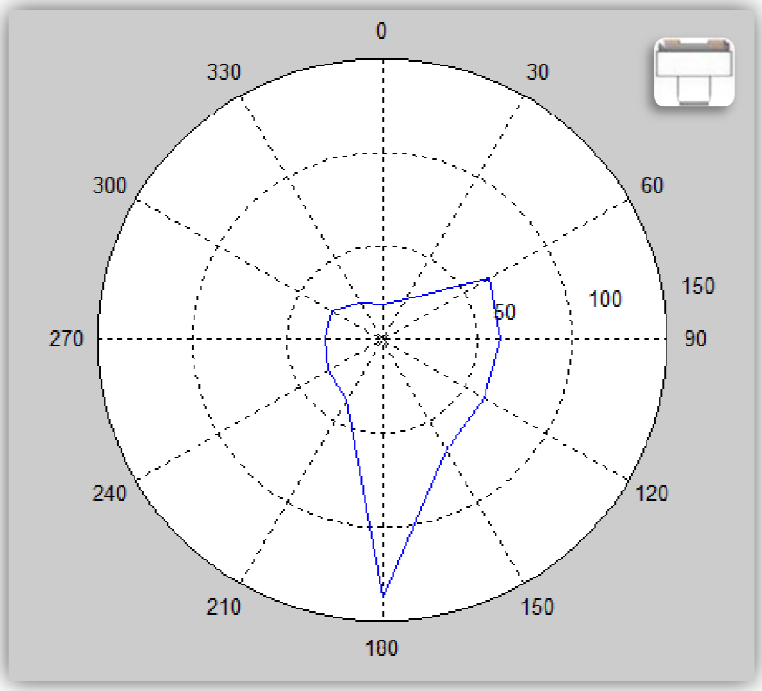


Figura 6.34: Diagrama polar posición 5

En la siguiente tabla, 6.5., se puede observar con claridad que la mayor distancia registrada corresponde a 180º. Como queremos evitar que el robot se dé media vuelta y deshaga el camino recorrido, este dato

se anulará como ya hicimos en la segunda posición, para que no sea tomada en cuenta a la hora de realizar el giro.

Como el rango de ángulos anulados es de 150º a 210º, obtenemos como mayor valor registrado los 65 cm a 60º. Nuevamente, como ya pasara en los casos anteriores, el robot realizará un giro en sentido horario de 90º (semiplano derecho).

Ángulo [º]	Distancia [cm]
0	19
30	25
60	65
90	62
120	62
150	68
180	137
210	37
240	33
270	30
300	31
330	23
360	19

Tabla 6.5: Relación ángulo-distancia posición 5

6.3.7.1. EVOLUCIÓN DE LA POBLACIÓN

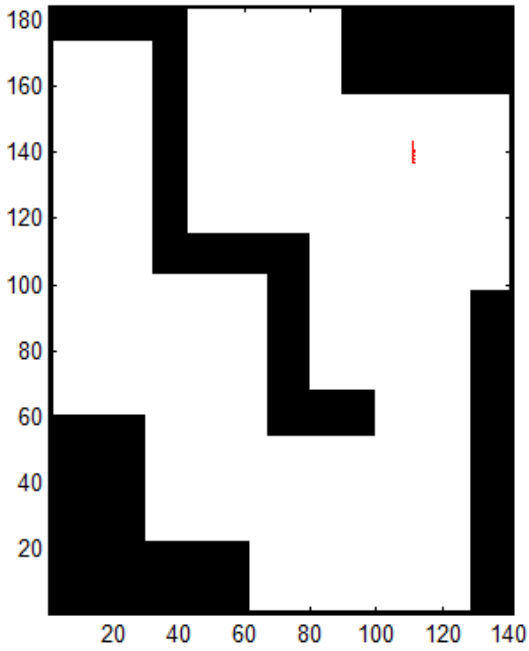


Figura 6.35: Evolución de la población con 10 iteraciones

Mejor individuo: [x = 111.29090, y = 139.80411, θ = 173.14254]

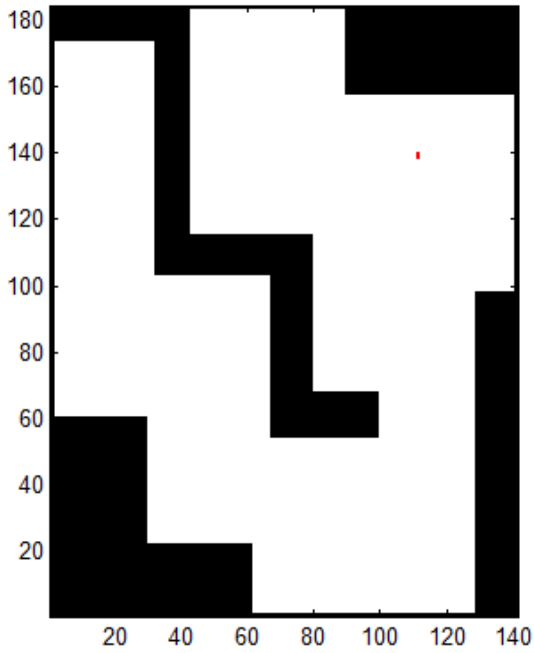


Figura 6.37: Evolución de la población con 30 iteraciones
Mejor individuo: [x = 111.68585, y = 138.85093, θ = 173.43882

Como ya sucedió para la posición 4, el algoritmo no necesita ni siquiera la mitad de las iteraciones preestablecidas inicialmente, ya que al llevar 60 iteraciones obtiene el resultado final como vemos en la siguiente figura, y más en detalle, en la figura 6.39.

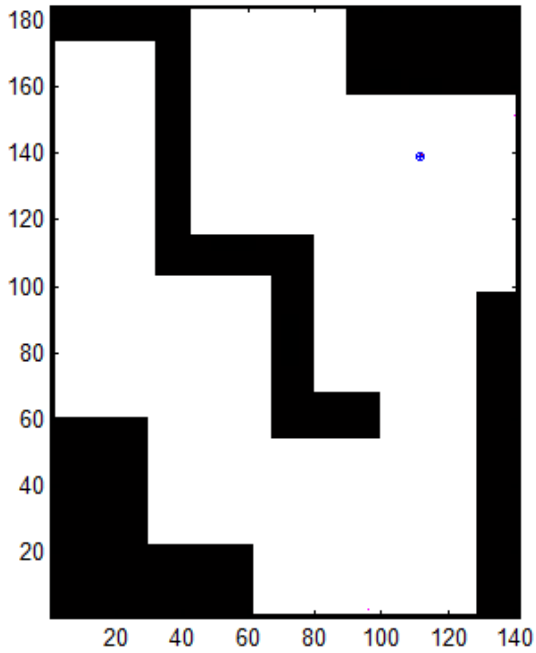


Figura 6.38: Evolución de la población con 60 iteraciones
Mejor individuo: [x = 111.68377, y = 138.84612, θ = 173.44487

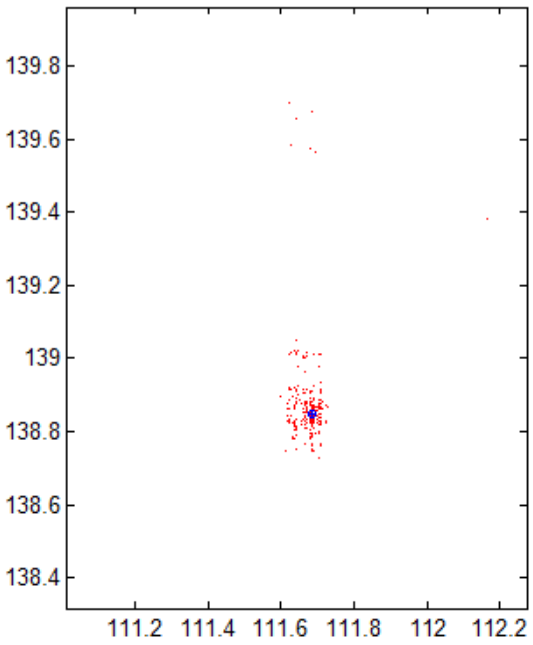


Figura 6.39: Detalle de la población con 60 iteraciones
Mejor individuo: [x = 111.68377, y = 138.84612, θ = 173.44487

6.3.8. SEXTA SECUENCIA

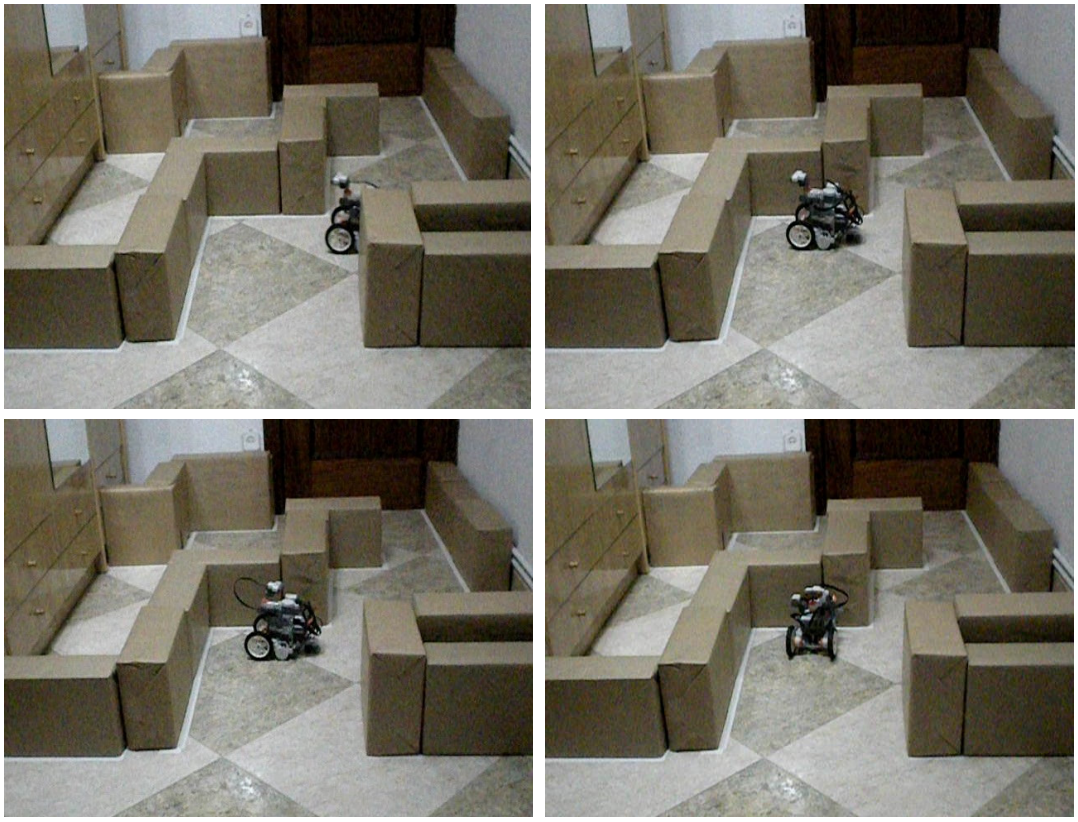


Figura 6.40: Arriba izquierda: Robot móvil avanza. Arriba derecha: Robot móvil se detiene.
Abajo izquierda: Robot móvil realiza chequeo del entorno. Abajo derecha: Robot móvil gira.

Como se puede observar en las imágenes de arriba, el robot móvil está cerca de concluir con éxito la "misión" de salir del laberinto; sin embargo, aún es necesario repetir la secuencia completa una vez más.

El robot avanzará hasta que el sensor ultrasonido detecte un obstáculo a menos de 20 cm, eso sucede, de acuerdo a la información de los sensores de los motores, cuando el robot ha recorrido 55,07 cm.

A continuación, se realizará un chequeo del entorno, del cuál obtendremos el siguiente diagrama polar:

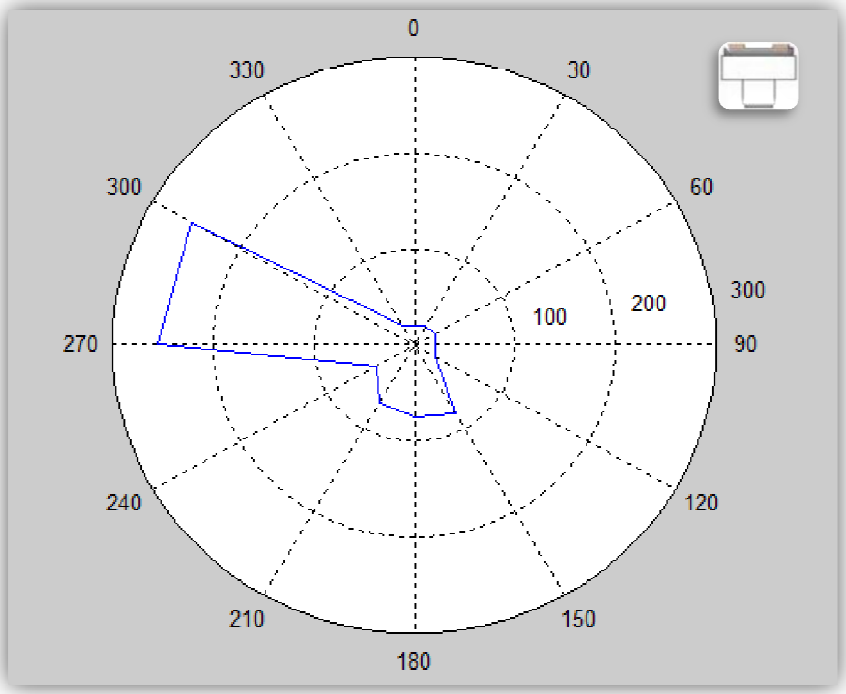


Figura 6.41: Diagrama polar posición 6

En el diagrama polar, figura 6.41, se puede observar una circunstancia que no había sucedido hasta ahora. La medida correspondiente a 270º y 330º es de 255 cm, éste represente el valor máximo de distancia que es capaz de medir el sensor ultrasonido y, se suele dar en dos casos: bien, cómo es lógico, cuando el valor medido es igual o superior a 255 cm, o bien, cuando la medida realizada es errónea. Nuestro caso, responde a la primera opción debido a que el robot se encuentra ante la salida.

Al encontrarse el valor máximo de distancia, 255 cm, en el semiplano izquierdo, el robot realizará un giro de 90º en sentido antihorario.

Ángulo [º]	Distancia [cm]
0	20
30	23
60	24
90	21
120	24
150	82
180	75
210	69
240	44
270	255
300	255
330	23
360	20

Tabla 6.6: Relación ángulo-distancia posición 6

6.3.8.1. EVOLUCIÓN DE LA POBLACIÓN

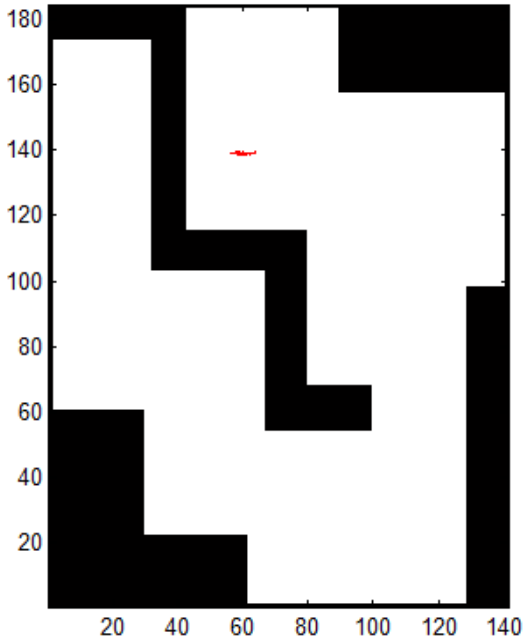


Figura 6.42: Detalle de la población con 10 iteraciones
Mejor individuo: [x = 60.72832, y = 138.84254, θ = 266.44779]

Nos encontramos ante la última ejecución del algoritmo de localización y, como se puede observar en la siguiente figura, solo se precisa de 40 iteraciones para alcanzar el resultado final.

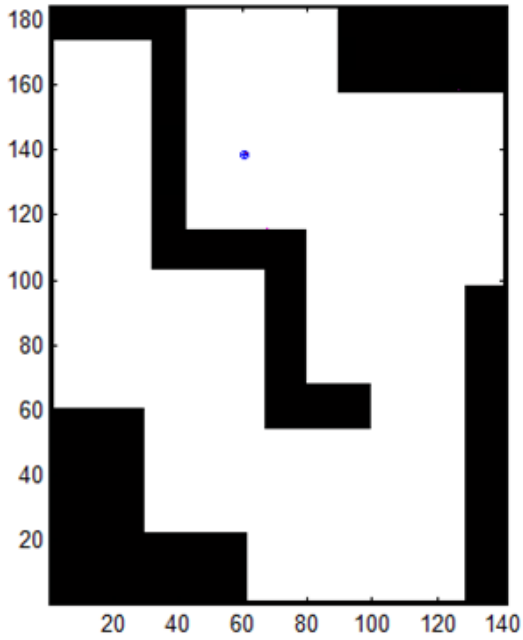


Figura 6.43: Detalle de la población con 40 iteraciones
Mejor individuo: [x = 60.69902, y = 138.76849, θ = 265.90156]

6.3.9. SALIDA DEL LABERINTO

Figura 6.44: Arriba izquierda: Robot móvil avanza. Arriba derecha: Robot sale del laberinto.

Al cumplirse la condición de salida, el robot móvil avanzará una distancia suficiente para encontrarse fuera de las lindes del laberinto, tal y como vemos en la figura 6.44., consiguiendo así su objetivo final.

Finalmente, para concluir su peripecia, se procede a la desactivación de todos los motores y sensores. Del mismo modo, se realiza la desconexión de la comunicación Bluetooth establecida con el PC.

7. CONCLUSIÓN

En este trabajo se han examinado a fondo dos cuestiones. En primer lugar, nos ocupa la capacidad para que un robot móvil de carácter educativo, que cuenta con sensores de bajo costo, sea capaz de esquivar los obstáculos que se encuentre a su paso y, en nuestro caso particular, encontrar la salida en un entorno laberíntico.

Para el control del robot, se ha descartado como medio de programación el software proporcionado por el fabricante. En su lugar, se ha utilizado Matlab que nos permite, por una parte, controlar el robot mediante una *toolbox* desarrollada en la Universidad de Aachen (Alemania) en la que vienen definidas un gran número de acciones para los motores y los sensores del robot y, por otra parte, Matlab, aporta una elevada capacidad de procesamiento matemático que permite la ejecución de aplicaciones de alto nivel, como es el algoritmo de localización utilizado en el presente proyecto.

Desde el punto de vista práctico hemos discutido la necesidad que tiene un robot móvil de auto-localizarse mientras navega a través de cualquier medio, para dotarle de un grado mayor de autonomía.

Al producirse el movimiento del vehículo, se puede estimar la nueva posición mediante sensores de odometría dispuestos en los motores de tracción

del robot. De este modo, se puede realizar el seguimiento de la posición, sin embargo, puesto que las medidas de estos sensores introducen errores, causados fundamentalmente por deslizamientos y las condiciones del terreno, la precisión de la estimación realizada irá disminuyendo progresivamente debido a que estos errores son acumulativos.

En este escenario, llegará un momento en el que el sistema no sepa con el grado de exactitud necesario su posición, lo que se soluciona haciendo uso de un método de localización. La localización es el proceso mediante el cuál se determinan la posición y orientación del robot utilizando la información procedente de los sensores externos.

En este proyecto se ha adaptado un método de localización basado en Differential Evolution y se aplica de forma global, lo que nos permite estimar la posición del robot sin la necesidad de tener información previa del estado inicial de éste.

Durante el proceso de localización, el robot tiene en cuenta la información proveniente de sus sensores internos que nos aportan información referente a la odometría, así como las medidas obtenidas por los sensores externos relativas al medio que le rodea.

En el capítulo 6, se realiza un análisis de forma secuencial de una ejecución del programa. En el contexto del presente proyecto, los resultados obtenidos se consideran bastante aceptables, más aún si, tenemos en cuenta que, dada la naturaleza del robot, trabajamos con una información proporcionada por unos sensores cuya exactitud, precisión o velocidad de respuesta es limitada, ya que se trata de transductores de bajo costo y, en consecuencia, sus características funcionales son más básicas y se ven condicionadas a mayores errores de lectura. No sin olvidar otras fuentes externas de errores como pueden ser el terreno por donde se desplaza el robot.

El potencial del algoritmo evolutivo diferencial utilizado queda de manifiesto en la localización del robot ejecutada cuando éste se encuentra en la posición inicial de partida, ya que es capaz de proporcionarnos una estimación precisa de la posición y orientación del robot, teniendo tan sólo en cuenta las 13 medidas realizadas por el sensor ultrasonido. A partir de este momento, el robot irá tomando decisiones en su estrategia de movimiento y realizará relocalizaciones relativas en función de los datos odométricos y del entorno cada vez que se encuentre detenido.

Podemos concluir afirmando que, los objetivos propuestos inicialmente para el presente proyecto, se alcanzan de forma satisfactoria.

7.1. TRABAJO FUTURO

Hasta aquí se ha probado la bondad y fiabilidad del sistema utilizando, para ello, un robot de carácter educativo dotado de unos sensores de bajo costo, por lo tanto, limitados en cuanto a características funcionales.

Una posibilidad futura, sería proponerlo como un método complementario de localización e implementarlo en un proyecto real donde el robot posea unos transductores con mayor precisión y con mejores prestaciones.

Dentro del marco de la localización se puede ir un paso más adelante e incluir técnicas SLAM (*Simultaneous Localization And Mapping)*, donde el robot móvil está presente en un entorno y posición desconocidos y es capaz de construir, incrementalmente, un mapa consistente del entorno, al tiempo que utiliza dicho mapa para determinar su propia localización.

Otra línea de investigación posible es la aplicación del Método *Fast Marching*, que permite el cálculo de trayectorias de forma optimizada en entornos complejos y otorga al robot móvil la capacidad de realizar giros, de forma segura, diferentes al cuarto de vuelta.

8. BIBLIOGRAFÍA

[1] J. González; A. Ollerro. *Estimación de la posición de un Robot Móvil*. Dpto. Departamento de Ingeniería de Sistemas y Automática, Universidad de Sevilla, vol. 29, nº 4, pp. 3-18, 1996.

[2] J. Borenstein; H. R. Everett; L. Feng. *Navigation Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Wellesley, MA, 1996.

[3] I. J. Cox; G. Wilfong. *Autonomous Robot Vehicles*. Springer-Verlag, New York, 1990.

[4] I. J. Cox. *Blanche: Position estimation for an autonomous robot vehicle.* IEEE Computer Society Press, vol. 2, pp.285-292, Los Alamitos, CA, 1991.

[5] A. Elfes. *Ocuppancy Grids: A Stochastic Spatial Representation for Active Robot Perception*. Proceedings of the Sixth Conference of Uncertainty in Artificial Intelligent, 1990.

[6] S. Thrun; M. Beetz; M. Bennewitz; W. Burgard; A. B. Cremers; F. Dellaert; D. Fox; D. Hähnel; C. Rosenberg; N. Roy; J.Schulte; D. Schulz. *Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva*. Journal of Robotics Research, vol. 19, nº 11, pp. 972-999, 2000.

[7] H. R. Everett. *Sensors for Mobile Robots: Theory and Application*. A. K. Peters, Ltd., Wellesley, MA, 1995.

[8] L. Feng; J. Borenstein; B. Everett. *Where am I? Sensors and Methods for Autonomous Mobile Robot Localization*. Technical Report UM-MEAM-94-21, The University of Michigan, 1994.

[9] J. Borenstein; L. Feng. *Measurement and Correction of Systematic Odometry Errors in Mobile Robots*. IEEE Journal of Robotics and Automation, vol. 12, nº 6, pp. 869-880, 1996.

[10] J. Kuipers; Y-T. Byun. *A robot exploration and mapping strategy basd on a semantic hierarchy of spatial represtations*. Journal of Robotics and Autonomous Systems 8, pp. 47-63, 1991.

[11] K. Chong; L. Kleeman. *Sonar based map building for mobile robot*. IEEE International Conference on Robotics and Automation, vol. 2, pp. 1700-1705, Alburquerque, New Mexico, 1997.

[12] J. Neira; J. Horn; J. Tardos; G. Schmidt. *Multisensor mobile robot localization*. IEEE International Conference on Robotics and Automation, pp. 673-679, Minneapolis, USA, 1996.

[13] W Burgard; A. Cremers; D. Fox; D. Hähnel; G. Lakemeyer; D. Schulz; W. Steiner; S. Thrun. *The interactive museum tour-guide robot*. Fifteenth National Conference on Artificial Intelligence (AAAI), 1998.

[14] H. Baltzaki; P. Trahanias. *Hybrid mobile robot localization using switching statespace models*. IEEE. International Conference on Robotics and Automation (ICRA), vol. 1, pp. 366-373, Washington DC, USA, 2002.

[15] J. Gutmann; C. Schegel. *Amos: Comparison of scan matching approaches for sel-localization in indoor enviroments*. First Euromicro Workshop on Advanced Mobile Robots, pp.61-67, 1996.

[16] F. Lu; E. Milios. *Globally consisten range scan alignment for environment mapping.* Department of Computer Science, York University, Ontario, Canadá, 1997.

[17] D. Fox; W. Burgard; S. Thrun; A.B. Cremers. *Position Estimation for Mobile Robots in Dynamic Environments*. Proceedings of the AAAI Fifteenth National Conference on Artificial Intelligence, 1998.

[18] J. García. *Sistema de posicionamiento y autolocalización para sillas de ruedas autónomas.* Tésis Doctoral, Departamento de Electrónica. Universidad de Alcalá, 2001.

[19] C. Cohen; F. Koss. *A comprehensive study of three object triangulation.* In proceedings of the SPIE Conference on Mobile Robots, Boston, MA, 1993.

[20] M. Betke; L. Gurvits. *Mobile robot localization using landmarks.* IEEE Transactions on Robotics and Automation, vol. 13, nº 2, pp. 251-263, 1997.

[21] S. Thrun; W. Burgard; D. Fox. *Probabilistic Robotic.* The MIT Press, Cambridge, MA, 2005.

[22] J. Borenstein; L. Feng. Umbmark. *A benchmark test for measuring odometry errors in mobile robots.* In Proceedings of the SPIE Conference on Mobile Robots, Philadelphia, USA, 1995.

[23] S. Thrun. *Probabilistic algorithms in robotics*. Technical Report CMU-CS-00-126, Computer Science Department. Carnegie Mellon University, Pittsburgh, PA., 2000.

[24] R. Kalman. *A new approach to linear filtering and prediction problems.* Transactions ASME Journal of Basic Engineering, 1960.

[25] P. S. Maybeck. *Stochastic models, estimation and control.* Academic Press, Inc., New York, USA, 1979.

[26] M. Grewal; A. Andrews. *Kalman filtering: theory and practice.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1993.

[27] R. Smith; P. Cheeseman. *On the estimation and representation of spatial uncertainty.* International Journal of Robotics Research, vol. 5, nº 4, 1987.

[28] G. Welch; G. Bishop. *An introduction to the Kalman Filter. Department of Computer Science,* University of North Carolina at Chapel Hill, NC, 2001.

[29] S. Thrun. *Particle Filters in Robotics.* In proceedings of Uncertainty in Intelligence Artificial, 2002.

[30] S. Garrido. *Identificación, Estimación y Control de Sistemas No-Lineales Mediante RGO*. Tesis Doctoral, Universidad Carlos III de Madrid, España, 2000.

[31] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, 1975.

[32] T. Blickle; L. Thiele. A *Comparison of Selection Schemes used in Evolutionary Algorithms*. Evolutionary Computation*, vol.* 4, nº 4, pp. 361-394, MIT-Press, 1996.

[33] K. A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems*. PhD Thesis, University of Michigan Ann Arbor, MI, USA, 1975.

[34] K. Price. *An introduction to Differential Evolution.* In New Ideas in Optimization, McGraw-Hill, London, 1999.

[35] R. Storn; K. Price. *Differential Evolution - A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces.* Technical Report TR-95-012, 1995.

[36] J. Lampinen. *A Bibliography of Differential Evolution Algorithm*. *Technical Report*, Department of Information Technology, Lappeenranta University of Technology, 2001.

[37] R. Storn; K. Price. *Minimizing the Real Functions of the ICEC'96 contest by Differential Evolution.* IEEE Conference on Evolutionary Computation, Nagoya, pp. 842 – 844, 1996.

[38] M. Gasperi; P. Hurbain; I. Hurbain. *Extending the Lego Mindstorms NXT to the next level.* Apress, New York, USA, 2007.

[39] J. Floyd. *Lego Mindstorms NXT: The Mayan Adventure*. Apress, New York, USA, 2006.

8.1. SITIOS WEB

[40] http://mindstorms.lego.com

[41] http://www.mindstorms.rwth-aachen.de/

[42] http://www.mathworks.com/matlabcentral/fileexchange/

9. APÉNDICE A

En este apartado se recogen las distintas subrutinas desarrollados para el control del robot móvil. La programación se ha realizado en MATLAB, haciendo uso, entre otros, de los comandos propios de la Toolbox RWTH-Lego Mindstorms versión 2.03.

❖ CONFIGURACIÓN BLUETOOTH

```
clear all
close all  %cerrar toda comunicación

disp('inicio configuracion');
COM_CloseNXT('all', 'bluetooth.ini');  %tomar archivo inicialización
handle = COM_OpenNXT('bluetooth.ini', 'check');
COM_SetDefaultNXT(handle);  %establecer comunicación
disp('fin configuracion')
```

❖ SUBRUTINA SEÑAL DE SALIDA

```
OpenSound(SENSOR_2, 'DB');
sound = GetSound(SENSOR_2);
sound=0; %descartar primera muestra

while sound<500
   OpenSound(SENSOR_2, 'DB');  %habilitar sensor sonido
   sound = GetSound(SENSOR_2);  %tomar muestras
end

CloseSensor(SENSOR_2);  %cerrar sensor sonido
NXT_PlayTone(1200,700);
```

❖ **SUBRUTINA CHEQUEO**

```
function [matriz,v_chequeo]=chequeo_robot(frec,rango)

%%Inicializamos variables
i=0; j=1;

%%Inicializamos Ultrasonido
OpenUltrasonic(SENSOR_1);

%% Matriz angulos-distancias
matriz=zeros(frec+1,2);

%% Posición inicial (0º grados)
dist=GetUltrasonic(SENSOR_1);
   matriz(j,1)=0;
   matriz(j,2)=dist;

%% Constantes motor
Rotacion = 33;
Ports =MOTOR_A;
V_Rotacion = 15;
V_Retorno = -40;

%% Inicializar motores
mRot1                   = NXTmotor(Ports(1));
mRot1.SpeedRegulation   = true;
mRot1.Power             = V_Rotacion;
mRot1.TachoLimit        = Rotacion;
mRot1.BrakeAtTachoLimit = true;

%% Rotación hasta 360º
for j = 1 : 12

   mRot1.SendToNXT();
   mRot1.WaitFor();

   dist=GetUltrasonic(SENSOR_1);
      j=j+1;
      i=i+1;

      matriz(j,2)=dist;
      matriz(j,1)=rango*i;
end

%% Retorno a posición inicial
inf=GetMotorSettings(MOTOR_A);
Retorno=inf.Angle;
```

```
mRot2 = NXTmotor(Ports(1)); mRot2.SpeedRegulation = true;
mRot2.Power = V_Retorno;
mRot2.TachoLimit = Retorno;
mRot2.BrakeAtTachoLimit = true;

mRot2.SendToNXT();
mRot2.WaitFor();

mRot1.Stop('off');
mRot2.Stop('off');

%% Diagrama Polar
figure(4);
polar(matriz(:,1)*pi/180, matriz(:,2));

%% Stop motores y sensor
StopMotor('all','off');
CloseSensor(SENSOR_1);

%% Vectores resultantes
v_polar=matriz(:,2); %Vector distancias
v_chequeo=v_polar(1:12); %Vector relocalización
```

- **SUBRUTINA AVANCE**

```
function odometry=avance_robot

%% Inicializamos motores
ResetMotorAngle(MOTOR_B);
ResetMotorAngle(MOTOR_C);

%% Habilitar ultrasonido
OpenUltrasonic(SENSOR_1);

%% Robot en marcha
SetMotor(MOTOR_B);
   SyncToMotor(MOTOR_C);
   SetPower(-32);
   SetAngleLimit(0);  %sin límite
   SetTurnRatio(0);  %recto
SendMotorSettings();
pause(0.5);  %tiempo de cortesía

%% Detection de pared
while GetUltrasonic(SENSOR_1) > 25
end
StopMotor('all', 'off');
NXT_PlayTone(1000, 800);
```

```
%% Inicializar motor y cerrar sensor
ResetMotorAngle(MOTOR_A);
CloseUltrasonic(SENSOR_1);

%% Odometría
B=GetMotorSettings(MOTOR_B);
C=GetMotorSettings(MOTOR_C);
average=-(B.Angle + C.Angle)/2;
odometry=(average/1350)*100; %1350 valor calibrado para 1m, resultado en cm.
```

- **SUBRUTINA GIRO**

```
function orientacion=giro_robot(frec,matriz)

%%Variables
h=0;

%% Obtención mayor distancia
c1=matriz(:,1);
c2=matriz(:,2);  %vector distancias que necesitamos para la relocalización.
c2(6:9)=0;  %descartamos la media vuelta

m_giro=[c1 c2];  %nueva matriz giro
M=max(m_giro);  %obtención de los máximos de la matriz
Dmax=M(1,2);  %distancia mayor

%% Constantes
QuarterTurnTicks = 135;  %calibración para giro de 90º
Ports = [MOTOR_B; MOTOR_C];  %motores sistema motriz
TurningSpeed = 20;

%% Obtención ángulo de giro
while h<=frec
   h=h+1;

   if Dmax==m_giro(h,2) %buscamos giro correspondiente a mayor distancia
      giro=m_giro(h,1);
         if giro > 180
            TurningSpeed=TurningSpeed*1; %giro antihorario
         else
            TurningSpeed=-TurningSpeed; %giro horario
         end
      break
   end
end

%% Inicializar motores
ResetMotorAngle(MOTOR_B);
ResetMotorAngle(MOTOR_C);
```

```
%% Constantes giro
mTurn1 = NXTmotor(Ports(2));
mTurn1.SpeedRegulation = true;
mTurn1.Power = TurningSpeed;
mTurn1.TachoLimit = QuarterTurnTicks;
mTurn1.BrakeAtTachoLimit = true;

 %% Ejecución del giro
mTurn2  = mTurn1;
mTurn2.Port  = Ports(1);
mTurn2.Power = - mTurn1.Power;

    mTurn1.SendToNXT();
    mTurn1.WaitFor();

    mTurn2.SendToNXT();
    mTurn2.WaitFor();

    NXT_PlayTone(1600,300);

mTurn1.Stop('off');
mTurn2.Stop('off');

%% Odometría giro
B=GetMotorSettings(MOTOR_B);
C=GetMotorSettings(MOTOR_C);

if giro>180  %giro en sentido antihorario
average=(abs(B.Angle) + abs(C.Angle))/2;
else
average=-(abs(B.Angle) + abs(C.Angle))/2;
end

orientacion=(average/135)*90;  %conversión a grados

%% Inicializar motores
ResetMotorAngle(MOTOR_B);
ResetMotorAngle(MOTOR_C);
```